



# The sip:carrier Handbook mr6.1.2

Sipwise GmbH  
<support@sipwise.com>

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	About this Handbook . . . . .	1
1.2	What is the sip:carrier? . . . . .	1
1.3	The Advantages of the sip:carrier . . . . .	1
1.4	Who is the sip:carrier for? . . . . .	2
1.5	Getting Help . . . . .	2
1.5.1	Phone Support . . . . .	2
1.5.2	Ticket System . . . . .	2
<b>2</b>	<b>System Architecture</b>	<b>3</b>
2.1	Hardware Architecture . . . . .	3
2.2	Component Architecture . . . . .	4
2.2.1	Provisioning . . . . .	5
2.2.2	Signaling and Media Relay . . . . .	8
2.2.3	Scaling beyond one Hardware Chassis . . . . .	10
2.2.4	Architecture for central core and local satellites . . . . .	11
<b>3</b>	<b>VoIP Service Administration Concepts</b>	<b>13</b>
3.1	Contacts . . . . .	13
3.2	Resellers . . . . .	13
3.3	SIP Domain . . . . .	14
3.3.1	Additional SIP Domains . . . . .	14
3.4	Contracts . . . . .	15
3.5	Customers . . . . .	15
3.5.1	Residential and SOHO customers . . . . .	15
3.5.2	Business customers with the Cloud PBX service . . . . .	16
3.5.3	SIP Trunking . . . . .	17
3.5.4	Mobile subscribers . . . . .	17

---

3.5.5	Pre-paid subscribers who use your calling cards . . . . .	17
3.6	Subscribers . . . . .	17
3.7	SIP Peerings . . . . .	18
<b>4</b>	<b>VoIP Service Configuration Scenario</b>	<b>20</b>
4.1	Creating a SIP Domain . . . . .	20
4.2	Creating a Customer . . . . .	21
4.3	Creating a Subscriber . . . . .	26
4.4	Domain Preferences . . . . .	30
4.5	Subscriber Preferences . . . . .	33
4.6	Creating Peerings . . . . .	34
4.6.1	Creating Peering Groups . . . . .	34
4.6.2	Creating Peering Servers . . . . .	36
4.6.3	Authenticating and Registering against Peering Servers . . . . .	45
4.7	Configuring Rewrite Rule Sets . . . . .	48
4.7.1	Inbound Rewrite Rules for Caller . . . . .	51
4.7.2	Inbound Rewrite Rules for Callee . . . . .	53
4.7.3	Outbound Rewrite Rules for Caller . . . . .	54
4.7.4	Outbound Rewrite Rules for Callee . . . . .	55
4.7.5	Emergency Number Handling . . . . .	55
4.7.6	Assigning Rewrite Rule Sets to Domains and Subscribers . . . . .	58
4.7.7	Creating Dialplans for Peering Servers . . . . .	59
4.7.8	Call Routing Verification . . . . .	59
<b>5</b>	<b>Features</b>	<b>65</b>
5.1	Managing System Administrators . . . . .	65
5.1.1	Configuring Administrators . . . . .	65
5.1.2	Access Rights of Administrators . . . . .	66
5.2	Access Control for SIP Calls . . . . .	69
5.2.1	Block Lists . . . . .	69

---

5.2.2	NCOS Levels	71
5.2.3	IP Address Restriction	78
5.3	Call Forwarding and Call Hunting	79
5.3.1	Setting a simple Call Forward	79
5.3.2	Advanced Call Hunting	79
5.4	Local Number Porting	86
5.4.1	Local LNP Database	86
5.4.2	External LNP via LNP API	89
5.5	Emergency Mapping	93
5.5.1	Emergency Mapping Description	94
5.5.2	Emergency Mapping Configuration	95
5.6	Emergency Priorization	101
5.6.1	Call-Flow with Emergency Mode Enabled	102
5.6.2	Configuration of Emergency Mode	105
5.6.3	Activating Emergency Mode	106
5.7	Header Manipulation	107
5.7.1	Header Filtering	107
5.7.2	Codec Filtering	108
5.7.3	Enable History and Diversion Headers	108
5.8	SIP Trunking with SIPconnect	109
5.8.1	User provisioning	109
5.8.2	Inbound calls routing	109
5.8.3	Number manipulations	109
5.8.4	Registration	112
5.9	Trusted Subscribers	113
5.10	Peer Probing	113
5.10.1	Introduction to Peer Probing Feature	113
5.10.2	Configuration of Peer Probing	114

5.10.3 Monitoring of Peer Probing . . . . .	116
5.10.4 Further Details for Advanced Users . . . . .	116
5.11 Fax Server . . . . .	117
5.11.1 Fax2Mail Architecture . . . . .	117
5.11.2 Sendfax and Mail2Fax Architecture . . . . .	118
5.12 Voicemail System . . . . .	119
5.12.1 Accessing the IVR Menu . . . . .	119
5.12.2 IVR Menu Structure . . . . .	120
5.12.3 Type Of Messages . . . . .	121
5.12.4 Folders . . . . .	122
5.12.5 Voicemail Languages Configuration . . . . .	122
5.12.6 Flowcharts with Voice Prompts . . . . .	123
5.13 Configuring Subscriber IVR Language . . . . .	128
5.14 Sound Sets . . . . .	128
5.14.1 Configuring Early Reject Sound Sets . . . . .	129
5.15 Conference System . . . . .	134
5.15.1 Configuring Call Forward to Conference . . . . .	135
5.15.2 Configuring Conference Sound Sets . . . . .	135
5.15.3 Joining the Conference . . . . .	137
5.15.4 Conference Flowchart with Voice Prompts . . . . .	137
5.16 Malicious Call Identification (MCID) . . . . .	139
5.16.1 Setup . . . . .	139
5.16.2 Usage . . . . .	140
5.16.3 Advanced configuration . . . . .	140
5.17 Subscriber Profiles . . . . .	140
5.17.1 Subscriber Profile Sets . . . . .	140
5.18 SIP Loop Detection . . . . .	143
5.19 Call-Through Application . . . . .	143

5.19.1 Administrative Configuration . . . . .	144
5.19.2 Call Flow . . . . .	146
5.20 Calling Card Application . . . . .	147
5.20.1 Administrative Configuration . . . . .	148
5.20.2 Call Flow . . . . .	150
5.21 Invoices and Invoice Templates . . . . .	151
5.21.1 Invoices Management . . . . .	151
5.21.2 Invoice Management via REST API . . . . .	153
5.21.3 Invoice Templates . . . . .	158
5.22 Email Reports and Notifications . . . . .	167
5.22.1 Email events . . . . .	167
5.22.2 Initial template values and template variables . . . . .	167
5.22.3 Password reset email template . . . . .	167
5.22.4 New subscriber notification email template . . . . .	168
5.22.5 Invoice email template . . . . .	168
5.22.6 Email templates management . . . . .	170
5.23 The Vertical Service Code Interface . . . . .	172
5.23.1 Vertical Service Codes for PBX customers . . . . .	172
5.23.2 Configuration of Vertical Service Codes . . . . .	173
5.23.3 Voice Prompts for Vertical Service Code Configuration . . . . .	173
5.24 Handling WebRTC Clients . . . . .	174
5.25 XMPP and Instant Messaging . . . . .	175
5.26 Call Recording . . . . .	175
5.26.1 Introduction to Call Recording Function . . . . .	175
5.26.2 Information on Files and Directories . . . . .	176
5.26.3 Configuration . . . . .	177
5.26.4 REST API . . . . .	181
5.27 SMS (Short Message Service) on Sipwise NGCP . . . . .	182

5.27.1 Configuration . . . . .	184
5.27.2 Monitoring, troubleshooting . . . . .	185
5.27.3 REST API . . . . .	192
<b>6 Customer Self-Care Interface and Menus</b>	<b>193</b>
6.1 The Customer Self-Care Web Interface . . . . .	193
6.1.1 Login Procedure . . . . .	193
6.1.2 Site Customization . . . . .	193
6.2 The Voicemail Menu . . . . .	199
<b>7 Billing Configuration</b>	<b>200</b>
7.1 Billing Profiles . . . . .	200
7.1.1 Creating Billing Profiles . . . . .	200
7.1.2 Creating Billing Fees . . . . .	202
7.1.3 Creating Off-Peak Times . . . . .	204
7.2 Peak Time Call Rating Modes . . . . .	206
7.2.1 Introduction to Call Rating Modes . . . . .	206
7.2.2 Typical Use Cases for Call Rating Modes . . . . .	207
7.2.3 Configuration of Call Rating Modes . . . . .	207
7.3 Prepaid Accounting . . . . .	207
7.4 Fraud Detection and Locking . . . . .	208
7.4.1 Fraud Lock Levels . . . . .	208
7.5 Billing Customizations . . . . .	209
7.5.1 Billing Networks . . . . .	210
7.5.2 Profile Mapping Schedule . . . . .	211
7.5.3 Profile Packages . . . . .	215
7.5.4 Vouchers . . . . .	226
7.5.5 Top-up . . . . .	229
7.5.6 Balance Overviews . . . . .	230
7.5.7 Usage Examples . . . . .	234

7.6	Billing Data Export	236
7.6.1	Glossary of Terms	237
7.6.2	File Name Format	237
7.6.3	File Format	238
7.6.4	File Transfer	250
<b>8</b>	<b>Provisioning REST API Interface</b>	<b>252</b>
8.1	API Workflows for Customer and Subscriber Management	252
8.2	API performance considerations	257
<b>9</b>	<b>Configuration Framework</b>	<b>258</b>
9.1	Configuration templates	258
9.1.1	.tt2 and .customtt2 files	258
9.1.2	.prebuild and .postbuild files	259
9.1.3	.services files	260
9.2	config.yml, constants.yml and network.yml files	261
9.3	ngccpcfg and its command line options	261
9.3.1	apply	261
9.3.2	build	261
9.3.3	commit	261
9.3.4	decrypt	262
9.3.5	diff	262
9.3.6	encrypt	262
9.3.7	help	262
9.3.8	initialise	262
9.3.9	pull	262
9.3.10	push	262
9.3.11	services	262
9.3.12	status	263



<b>10 Network Configuration</b>	<b>264</b>
10.1 General Structure	264
10.1.1 Available Host Options	265
10.1.2 Interface Parameters	266
10.2 Advanced Network Configuration	267
10.2.1 Extra SIP Sockets	267
10.2.2 Extra SIP and RTP Sockets	268
10.2.3 Alternative RTP Interface Selection Using ICE	271
10.2.4 Extended RTP Port Range Using Multiple Interfaces	271
10.2.5 Cluster Sets	272
<b>11 Licenses</b>	<b>276</b>
11.1 What is Subject to Licensing?	276
11.2 How Licensing Works	276
11.3 How to Configure Licenses	277
11.4 How to Monitor License Client	277
<b>12 Software Upgrade</b>	<b>278</b>
12.1 Release Notes	278
12.2 Overview	278
12.3 Planning a software upgrade	279
12.4 Preparing the software upgrade	279
12.4.1 Log into the standby management server (web01a/db01a).	279
12.4.2 Log into the remaining servers	281
12.5 Upgrading the sip:carrier	282
12.5.1 Preparing for maintenance mode	282
12.5.2 Upgrading ONLY the first standby management node "A" (web01a/db01a)	283
12.5.3 Upgrading the standby database node "A" (db*a)	284
12.5.4 Upgrading other standby nodes "A" (lb*a/prx*a)	284
12.5.5 Promote ALL standby nodes "A" to active.	285

12.5.6 Upgrading ALL standby nodes "B" (web*b/db*b/lb*b/prx*b)	285
12.6 Post-upgrade steps	285
12.6.1 Disabling maintenance mode	285
12.6.2 Post-upgrade checks	286
12.7 Applying the Latest Hotfixes	286
12.7.1 Update the approx cache on the <b>standby</b> management node	286
12.7.2 Apply hotfixes on the <b>standby</b> management node	287
12.7.3 Apply hotfixes on all other <b>standby</b> nodes	287
12.7.4 Promote the <b>standby</b> nodes to active	287
12.7.5 Apply hotfixes on new standby nodes	287
<b>13 Backup, Recovery and Database Maintenance</b>	<b>288</b>
13.1 sip:carrier Backup	288
13.1.1 What data to back up	288
13.1.2 The built-in backup solution	288
13.2 Recovery	289
13.3 Reset Database	289
13.4 Accounting Data (CDR) Cleanup	289
13.4.1 Cleanuptools Configuration	290
13.4.2 Accounting Database Cleanup	290
13.4.3 Exported CDR Cleanup	293
<b>14 Platform Security, Performance and Troubleshooting</b>	<b>294</b>
14.1 Sipwise SSH access to sip:carrier	294
14.2 Firewalling	294
14.2.1 Firewall framework	294
14.2.2 NGCP firewall configuration	296
14.2.3 IPv4 System rules	296
14.2.4 Custom rules	300
14.2.5 Example firewall configuration section	300

14.3 Password management . . . . .	301
14.3.1 The "root" account . . . . .	301
14.3.2 The "administrator" account . . . . .	301
14.3.3 The "cdreexport" account . . . . .	302
14.3.4 The MySQL "root" user . . . . .	302
14.3.5 The "ngcpsoap" account . . . . .	302
14.4 SSL certificates. . . . .	302
14.5 Securing your sip:carrier against SIP attacks . . . . .	303
14.5.1 Denial of Service . . . . .	303
14.5.2 Bruteforcing SIP credentials . . . . .	304
14.6 Topology Hiding . . . . .	305
14.6.1 Introduction to Topology Hiding on NGCP . . . . .	305
14.6.2 Configuration of Topology Hiding . . . . .	305
14.6.3 Considerations for Topology Hiding . . . . .	306
14.7 System Requirements and Performance . . . . .	306
14.8 Troubleshooting . . . . .	308
14.8.1 Collecting call information from logs . . . . .	311
14.8.2 Collecting SIP traces . . . . .	312
<b>15 Monitoring and Alerting</b> . . . . .	<b>313</b>
15.1 Internal Monitoring . . . . .	313
15.1.1 Process monitoring via monit . . . . .	313
15.1.2 System monitoring via Telegraf . . . . .	313
15.1.3 NGCP-specific monitoring via ngcp-witnessd . . . . .	313
15.1.4 Monitoring data in InfluxDB . . . . .	314
15.2 Statistics Dashboard . . . . .	314
15.3 External Monitoring Using SNMP . . . . .	315
15.3.1 Overview and Initial Setup . . . . .	315
15.3.2 Details . . . . .	315

<b>16 Extensions and Additional Modules</b>	<b>321</b>
16.1 Cloud PBX	321
16.1.1 Configuring the Device Management	321
16.1.2 Preparing PBX Rewrite Rules	330
16.1.3 Creating Customers and Pilot Subscribers	334
16.1.4 Creating Regular PBX Subscribers	344
16.1.5 Assigning Subscribers to Devices	350
16.1.6 Configuring Sound Sets for the Customer PBX	356
16.1.7 Auto-Attendant Function	358
16.1.8 Cloud PBX Groups with Busy Members	364
16.1.9 Configuring Call Queues	366
16.1.10 Device Auto-Provisioning Security	368
16.1.11 Device Bootstrap and Resync Workflows	370
16.1.12 Device Provisioning and Deployment Workflows	378
16.1.13 List of available pre-configured devices	381
16.1.14 Phone features	385
16.1.15 Shared line appearance	417
16.2 Sipwise sip:phone App (SIP client)	417
16.2.1 Zero Config Launcher	418
16.2.2 Mobile Push Notification	422
16.3 Lawful Interception	443
16.3.1 Introduction	443
16.3.2 Architecture and Configuration of LI Service	445
16.3.3 X1, X2 and X3 Interface Specification	452
16.4 3rd Party Call Control	466
16.4.1 Introduction	466
16.4.2 Details of Call Processing with PCC	467
16.4.3 Voicemail Notification	473

16.4.4 Incoming Short Message Acceptance . . . . .	475
16.4.5 Configuration of PCC . . . . .	477
16.4.6 Troubleshooting of PCC . . . . .	478
<b>A Basic Call Flows</b>	<b>482</b>
A.1 General Call Setup . . . . .	482
A.2 Endpoint Registration . . . . .	483
A.3 Basic Call . . . . .	486
A.4 Session Keep-Alive . . . . .	487
A.5 Voicebox Calls . . . . .	488
<b>B NGCP configs overview</b>	<b>490</b>
B.1 config.yml Overview . . . . .	490
B.1.1 apps . . . . .	490
B.1.2 asterisk . . . . .	490
B.1.3 autoprov . . . . .	492
B.1.4 backuptools . . . . .	492
B.1.5 bootenv . . . . .	493
B.1.6 cdrexport . . . . .	494
B.1.7 checktools . . . . .	495
B.1.8 cleanuptools . . . . .	497
B.1.9 cluster_sets . . . . .	498
B.1.10 database . . . . .	499
B.1.11 faxserver . . . . .	499
B.1.12 general . . . . .	499
B.1.13 haproxy . . . . .	500
B.1.14 heartbeat . . . . .	500
B.1.15 intercept . . . . .	500
B.1.16 kamailio . . . . .	501
B.1.17 Inpd . . . . .	512

---

B.1.18 mediator	513
B.1.19 modules	513
B.1.20 nginx	513
B.1.21 ntp	514
B.1.22 ossbss	514
B.1.23 pbx (only with additional cloud PBX module installed)	516
B.1.24 prosody	516
B.1.25 pushd	516
B.1.26 qos	519
B.1.27 rate-o-mat	519
B.1.28 redis	519
B.1.29 reminder	520
B.1.30 rsyslog	520
B.1.31 rtpproxy	521
B.1.32 security	523
B.1.33 sems	524
B.1.34 sms	525
B.1.35 snmpagent	527
B.1.36 sshd	527
B.1.37 sudo	527
B.1.38 voisniff	527
B.1.39 www_admin	529
B.2 constants.yml Overview	532
B.3 network.yml Overview	532
<b>C NGCP-Faxserver Configuration</b>	<b>536</b>
C.1 Faxserver Components	536
C.2 Enabling Faxserver	536
C.3 Fax Templates Configuration	537

C.4 Fax Services Configuration per Subscriber . . . . .	537
C.5 Fax2Mail and SendFax Settings . . . . .	538
C.6 Mail2Fax Settings . . . . .	539
C.7 Sending Fax from Web Panel . . . . .	541
C.8 Faxserver Mail2Fax Configuration . . . . .	542
C.9 Sending Fax Using E-mail Clients . . . . .	542
C.10 Managing Faxes via the REST API . . . . .	543
C.10.1 Configuring Fax Settings . . . . .	543
C.10.2 Sending a Fax . . . . .	544
C.10.3 Receiving a Fax . . . . .	544
C.10.4 Configuring Mail2Fax Settings . . . . .	545
C.10.5 Using Advanced Faxserver and Mail2Fax Settings via the REST API . . . . .	546
C.11 Troubleshooting . . . . .	547
C.11.1 Session ID (SID) . . . . .	547
C.11.2 Fax Storage Location . . . . .	548
<b>D RTC:engine</b>	<b>549</b>
D.1 Overview . . . . .	549
D.2 RTC:engine enabling . . . . .	549
D.2.1 Enabling services via CLI . . . . .	549
D.2.2 Enabling via Panel for resellers and subscribers . . . . .	550
D.2.3 Create RTC:engine session . . . . .	550
D.3 RTC:engine protocol details . . . . .	551
D.3.1 Terminology . . . . .	551
D.3.2 Messages . . . . .	552
D.3.3 Account . . . . .	554
D.3.4 Call . . . . .	558
D.3.5 Session . . . . .	564
<b>E comx-fileshare-service</b>	<b>566</b>

---

E.1	Overview	566
E.2	Configuration and Usage	566
E.2.1	Change authentication method	566
E.2.2	Database Structure	567
E.3	Activation of Filesharing Service on NGCP	568
E.4	Message Sequence Chart	569
E.4.1	Simple Message Sequence	569
E.4.2	Detailed Message Sequence	570
E.5	API of Filesharing Service	570
E.5.1	HTTP Authentication	570
E.5.2	Upload and Download with Simple Identification	571
E.5.3	Upload and Download with Session Identification	571
E.5.4	Curl Example for Simple Upload Request	571
E.5.5	Upload Parameters	571
E.5.6	Number of Possible Downloads	573
<b>F</b>	<b>NGCP Internals</b>	<b>574</b>
F.1	Pending reboot marker	574
F.2	Redis id constants	574
F.2.1	InfluxDB monitoring keys	575
F.3	Enum preferences	576



# 1 Introduction

## 1.1 About this Handbook

This handbook describes the architecture and the operational steps to install, operate and modify the Sipwise sip:carrier.

In various chapters, it describes the system architecture, the installation and upgrade procedures and the initial configuration steps to get your first users online. It then dives into advanced preference configurations such as rewrite rules, call blockings, call forwards, etc.

There is a description of the customer self-care interface, how to configure the billing system and how to provision the system via the provided APIs.

Finally, it describes the internal configuration framework, the network configuration and gives hints about tweaking the system for security and performance.

## 1.2 What is the sip:carrier?

The sip:carrier is a SIP based Open Source Class5 VoIP soft-switch platform providing rich telephony services. It offers a wide range of features to end users (call forwards, voicemail, conferencing, call blocking, click-to-dial, call-lists showing near-realtime accounting information, etc.), which can be configured by them using the customer-self-care web interface. For operators, it offers a fully web-based administrative panel, allowing them to configure users, peerings, billing profiles, etc., as well as viewing real-time statistics of the system. For tight integration into existing infrastructures, it provides a powerful REST API.

The sip:carrier comes pre-installed on six+ servers in one+ IBM Flex Chassis, see Section 2. Apart from your product specific configuration, there is no initial configuration or installation to be done to get started.

## 1.3 The Advantages of the sip:carrier

Opposed to other free VoIP software, the sip:carrier is not a single application, but a whole software platform, the Sipwise NGCP (Sipwise Next Generation Communication Platform), which is based on Debian GNU/Linux.

Using a highly modular design approach, the NGCP leverages popular open-source software like MySQL, NGINX, Kamailio, SEMS, Asterisk, etc. as its core building blocks. These blocks are glued together using optimized and proven configurations and workflows and are complemented by functionality developed by Sipwise to provide fully-featured and easy to operate VoIP services.

The installed applications are managed by the NGCP Configuration Framework, which makes it possible to change system parameters in a single place, so administrators don't need to have any knowledge of the dozens of different configuration files of the different packages. This provides a very easy and bullet-proof way of operating, changing and tweaking the otherwise quite complex system.

Once configured, integrated web interfaces are provided for both end users and administrators to use the sip:carrier. By using the provided provisioning and billing APIs, it can be integrated tightly into existing OSS/BSS infrastructures to optimize workflows.

## 1.4 Who is the sip:carrier for?

The sip:carrier is specifically tailored to companies who want to provide fully-featured SIP-based VoIP service without having to go through the steep learning curve of SIP signalling, integrating the different building blocks to make them work together in a reasonable way. The sip:carrier is already deployed all around the world by all kinds of VoIP operators, using it as Class5 soft-switch, as Class4 termination platform or even as Session Border Controller with all kinds of access networks, like Cable, DSL, WiFi and Mobile networks.

## 1.5 Getting Help

### 1.5.1 Phone Support

Depending on your support contract, you are eligible to contact our Support Team by phone either in business hours or around the clock. Business hours refer to the UTC+1 time zone (Europe/Vienna). Please check your support contract to find out the type of support you've purchased.

Before calling our Support Team, please also open a ticket in our Ticket System and provide as much detail as you can for us to understand the problems, fix them and investigate the cause. Please provide the number of your newly created ticket when asked by our support personnel on the phone.

You can find phone numbers, Ticket System URL, and account information in your support contract. Please make this information available to the persons in your company maintaining the sip:carrier.

### 1.5.2 Ticket System

Depending on your support contract, you can create either a limited or an unlimited amount of support tickets on our Web-based Ticket System. Please provide as much information as possible when opening a ticket, especially the following:

- **WHAT** is affected (e.g. the whole system is unreachable, or customers can't register or place calls)
- **WHO** is affected (e.g. all customers, only parts of it, and **WHICH** parts - only customers in a particular domain or customers with specific devices, etc.)
- **WHEN** did the problem occur (time frames, or after the firmware of specific devices types have been updated, etc.)

Our Support Team will ask further questions via the Ticket System along the way of troubleshooting your issue. Please provide the information as soon as possible to solve your issue promptly.

## 2 System Architecture

### 2.1 Hardware Architecture

The sip:carrier starts with a minimum deployment of 50,000 subscribers, requiring one chassis with two web servers, two db servers, two load balancers and two proxies. A fully deployed sip:carrier for 200,000 subscribers fills the chassis up with 14 servers, containing two web servers, two db servers, two load balancers and 8 proxies.



Figure 1: Hardware setup for single chassis

The system is based on an IBM Flex Chassis taking up rack space of 10U with 14 computing nodes based on IBM x240 servers.

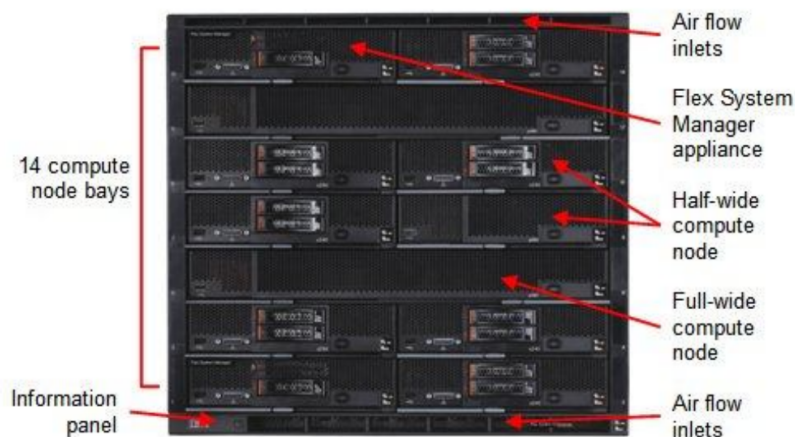


Figure 2: Chassis front view

All nodes are equipped equally with two hard disks in Raid-1 mode.

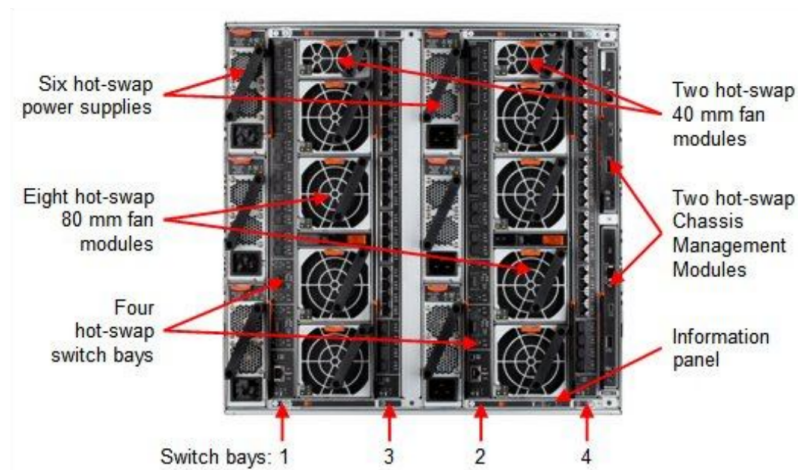


Figure 3: Chassis back view

The power supply is designed fully redundant in an N+N fashion with N=3, for example to feed 3 PSUs with normal power and 3 PSUs with UPS power.



Figure 4: Chassis switch module

Each chassis is equipped with two EN2092 Gigabit Ethernet switches providing 10 1GbE uplinks each. Four 10GbE uplinks are optional and need to be licensed separately if needed.

## 2.2 Component Architecture

The sip:carrier is composed by a cluster of four different node types, which are all deployed in active/standby pairs:

- **Web-Servers** (web1a/web1b): Provide northbound interfaces (CSC, API) via HTTPS for provisioning
- **DB-Servers** (db1a/db1b): Provide the central persistent SQL data store for customer data, peering configuration, billing data etc.
- **Proxy-Servers** (proxy1a/proxy1b .. proxy4a/proxy4b): Provide the SIP and XMPP signalling engines, application servers and media relays to route Calls and IM/Presence and serve media to the endpoints.

- **Load-Balancers** (lb1a/lb1b): Provide a perimeter for SIP and XMPP signalling.

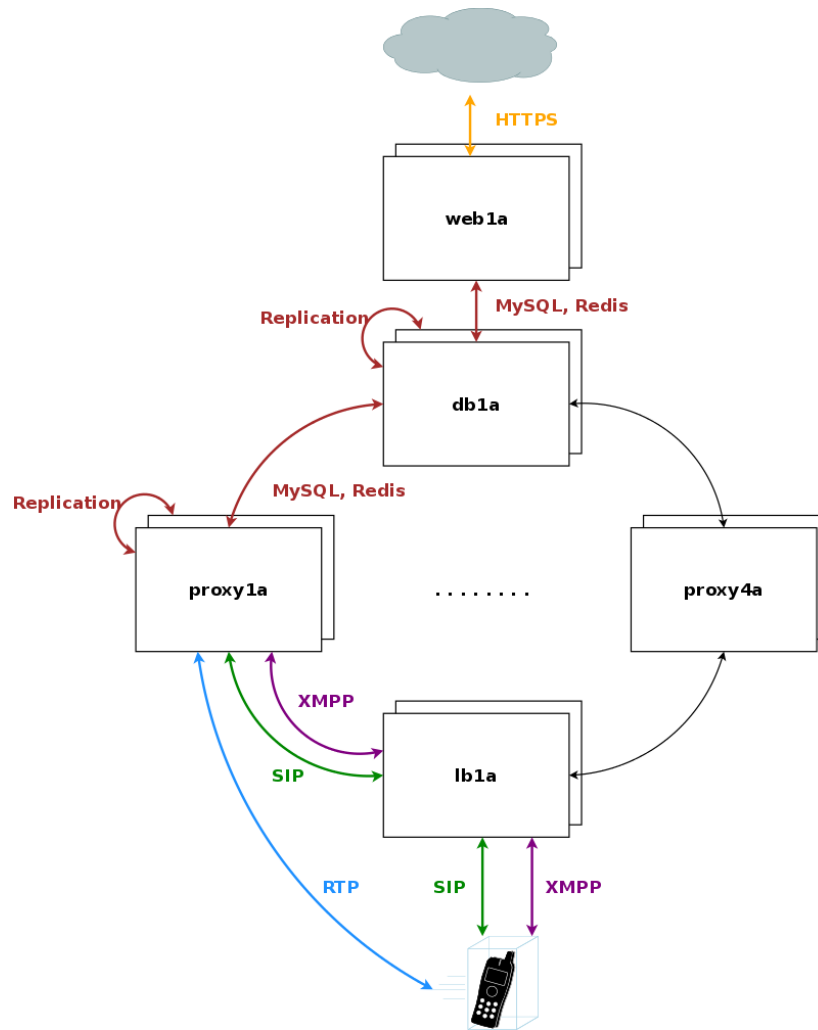


Figure 5: Architecture Overview

The system is provisioned via the web servers on a central pair of db servers. Signalling is entering the system via the lb servers to a cluster of proxies, which in turn communicate directly (caching and shared data) and indirectly (static provisioning data replicated via master/slave) with the db servers. Each pair of proxy is capable of handling any subscriber, so subscribers are not bound to specific "home proxies". Once a call starts on a proxy pair, it is ensured that the full range of services is provided on that pair (voicemail, media, billing, . . .) until call-teardown. Failures on an active proxy node cause a fail-over to the corresponding stand-by node within the proxy pair, taking over the full signalling and media without interruptions.

### 2.2.1 Provisioning

Any HTTPS traffic for provisioning (web interfaces, northbound APIs) but also for phone auto-provisioning enters the platform on the active web server. The web server runs an nginx instance acting as a reverse proxy for the ngcp-panel process, which in turn provides the provisioning functionality.

The web server is connected to the db server pair, which provides a persistent relational data store via MySQL and a high-performance system cache using Redis key-value store.

### 2.2.1.1 API and Web Interface

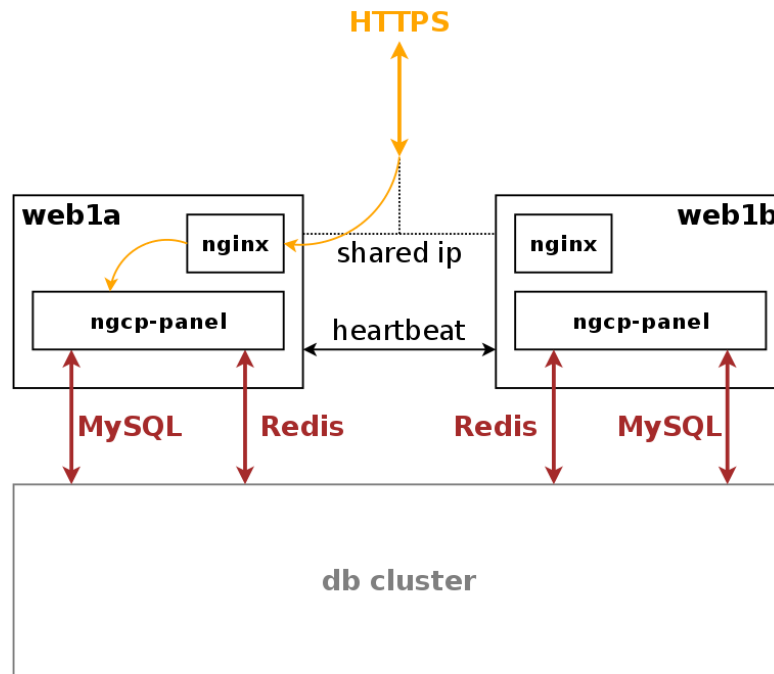


Figure 6: Web Server Overview

The web server pair is an active/standby pair of nodes connected via heartbeat. If one of the servers fail (by losing connection to the outside while the standby server is still connected, or caused by a hardware failure, or if it's down due to maintenance), the standby server takes over the shared IP address of the active node and continues serving the provisioning interface.

### 2.2.1.2 Provisioning Database

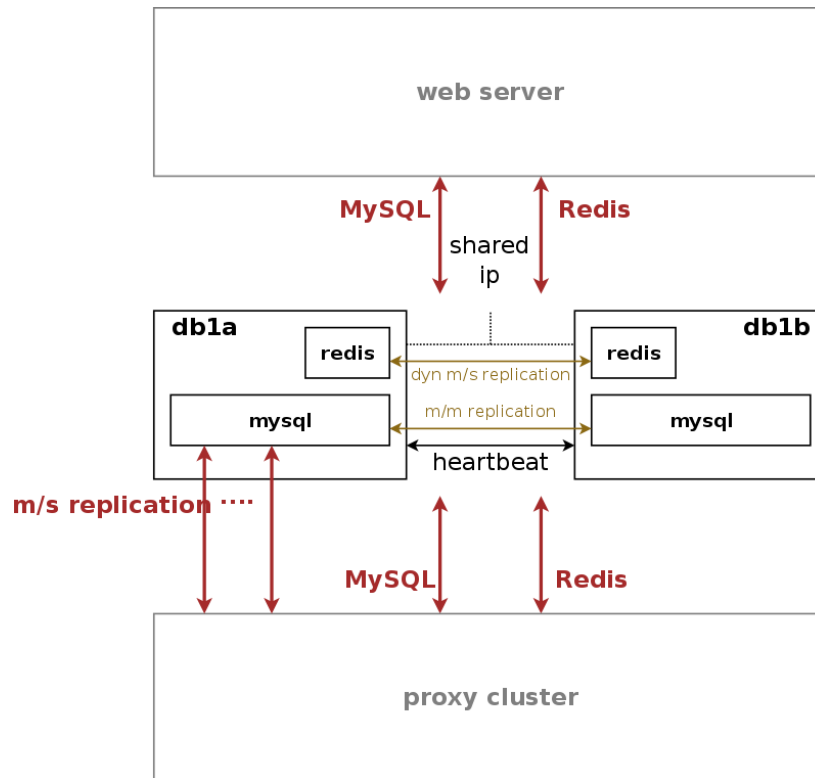


Figure 7: DB Server Overview

The db server pair is another active/standby pair with automatic fail-over. Nodes in the pair are running a MySQL master/master replication with replication integrity checks to ensure data redundancy and safety. Any changes via provisioning interfaces are stored in the MySQL cluster. The second service is a redis master/slave replication with automatic master propagation on fail-over. This redis cluster is used as a high-performance volatile system cache for various components which need to share state information across nodes.

### 2.2.1.3 Persistent MySQL Database

The MySQL instances on the db nodes synchronize via row-based master/master replication. In theory, any of the two servers in the pair can be used to write data to the database, however in practice a shared IP is used towards clients accessing the service, so only one node will receive the write requests. This is done to ensure transparent and instant convergence of the db cluster on fail-over for the clients.

On top of that, the first node of the db pair also acts as a master in a master/slave replication towards all proxy nodes in the system. That way, proxies can access read-only provisioning data directly from their local databases, resulting in reduced latency and significant off-loading of read queries on the central db cluster.

### 2.2.1.4 Central Redis Cache

A redis master/slave setup is used to provide a high-performance key/value storage for global system data shared across proxies. This includes concurrent call counters for customers and subscribers, as a subscriber could place two simultaneous calls via two different proxy pairs.

## 2.2.2 Signaling and Media Relay

Any signalling traffic enters and leaves the system via load balancers, which act as a perimeter towards the customer devices and performs NAT handling, DoS and DDoS mitigation. New connections are routed to a random pair of proxy servers, which do the actual routing for SIP and XMPP. The proxy servers also engage media relays for voice and video streams, which bypass the load balancers and communicate directly with the customer devices for performance reasons.

### 2.2.2.1 Load Balancing of Signalling

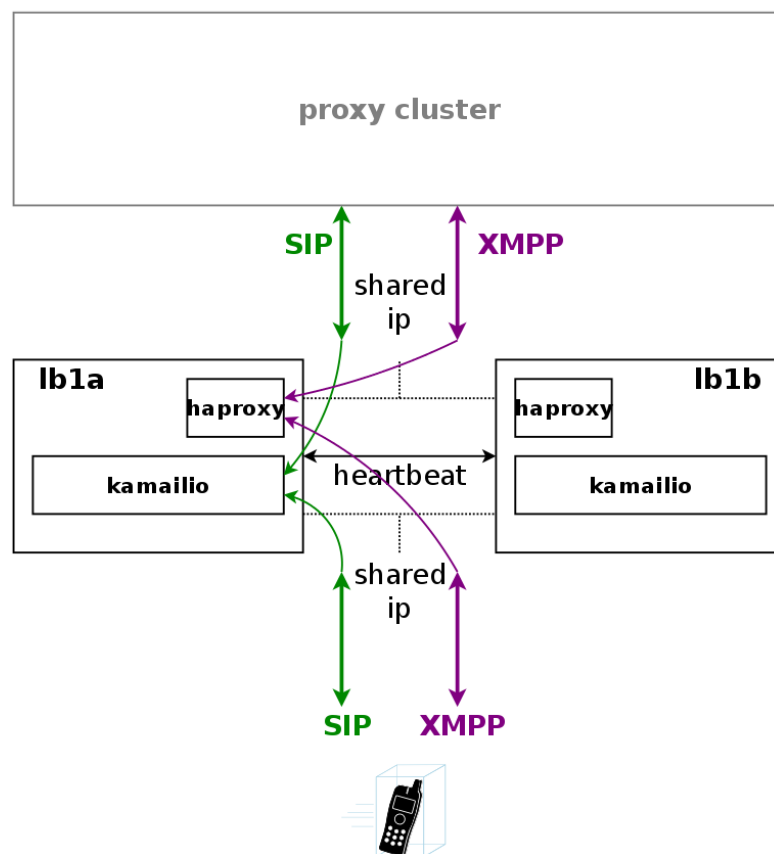


Figure 8: Load Balancer Overview

A node in a load balancer pair runs two services besides the usual heartbeat.

One is a state-less instance of kamailio, providing an extremely fast relay of SIP messages. Kamailio takes care of converting



TCP and TLS connections from the customer devices to UDP for internal communication towards proxies, and it performs far-end NAT traversal by inspecting the SIP messages and comparing it to the actual source address where packets have been received from, then modifying the SIP messages accordingly. If a SIP message is received by the load balancer, it distinguishes between new and ongoing SIP transactions by inspecting the To-Tags of a message, and it determines whether the message is part of an established dialog by inspecting the Route header. Sanity checks are performed on the headers to make sure the call flows adhere to certain rules for not being able to bypass any required element in the routing path. In-dialog messages are routed to the corresponding proxy servers according to the Route defined in the message. Messages initiating a new transaction and/or dialog (registrations, calls etc) are routed to a randomly selected proxy. The selection algorithm is based on a hash over the Call-ID of the message, so the same proxy sending a authentication challenge to an endpoint will receive the authenticated message again.

The second service running on a load balancer is haproxy, which is acting as load balancing instance for XMPP messages. The same way the SIP load balancer routes SIP messages to the corresponding proxy, the haproxy passes XMPP traffic on to the proxy maintaining a session with a subscriber, or randomly selects a proxy in case of a new connection while automatically failing over on timeouts.

### 2.2.2.2 Message Routing and Media Relay

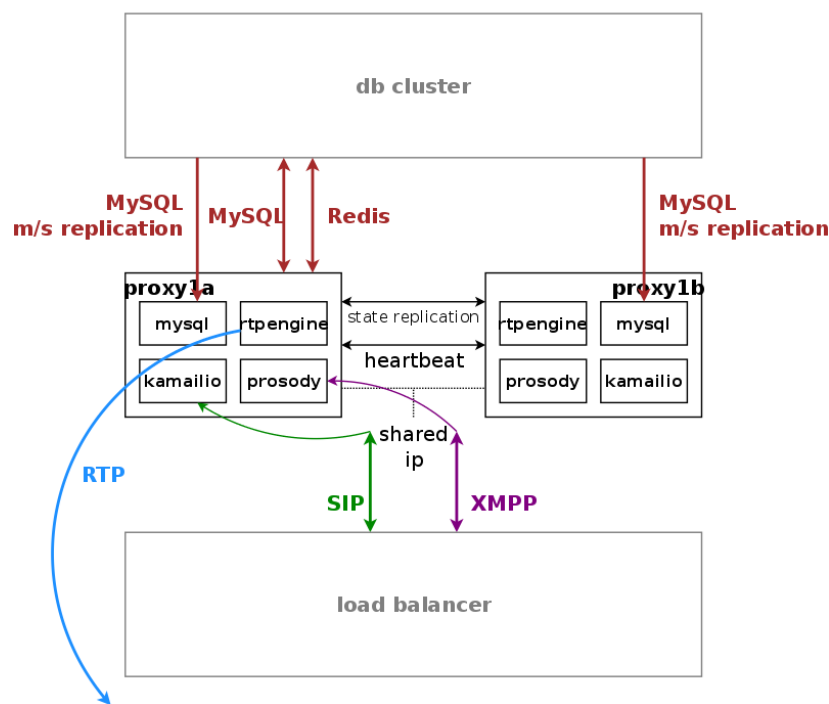


Figure 9: Proxy Server Overview

Proxy servers also come in pairs, and by default there are four pairs of proxies in a standard sip:carrier setup.

The proxies are responsible for doing the actual SIP routing and media handling and the XMPP presence and chat message deliveries. Each proxy pair can handle any subscriber on the overall system, compared to the concept of "home proxies" in other architectures. The advantage of this approach is that the overall system can be scaled extremely easily by adding more proxy pairs without having to redistribute subscribers.

Once a load balancer sends a new message to a proxy, the SIP transaction and/or dialog gets anchored to this proxy. That way it is ensured that a call starting on a proxy is also ended on the same proxy. Hence, the full range of feature handling like media relay, voicemail, fax, billing and rating is performed on this proxy. So, there is no a central point for various tasks, potentially leading to a non-scalable bottleneck. Due to the anchoring, proxies come in pairs and replicate all internal state information to the standby node via redis. In case of fail-over, the full signalling and media are moved to the standby node without interruption.

The complete static subscriber information like authentication credentials, number mappings, feature settings etc. are replicated from the db cluster down to the local MySQL instance of the proxies. The ratio of db read requests of static subscriber data versus reading and writing volatile and shared data is around 15:1, and this approach moves the majority of the static read operations from the central db cluster to the local proxy db.

Volatile and shared information needed by all proxies in the cluster is read from and written to the db cluster. This mainly includes SIP registration information and XMPP connection information.

Billing and rating is also performed locally on the proxies, and only completed CDRs (rated or unrated depending on whether rating is enabled) are transferred to the central db cluster for consumption via the northbound interfaces.

For SIP, the relevant instances on a proxy are kamailio acting as a stateful proxy for SIP registration and call routing, seds acting as a back-to-back user-agent for prepaid billing and application server, rtpengine as media relay and RTP/SRTP transcoder, and asterisk as voicemail server. XMPP is handled by an instance of prosody, and several billing processes mediate start and stop records into CDRs and rate them according to the relevant billing profiles.

### 2.2.3 Scaling beyond one Hardware Chassis

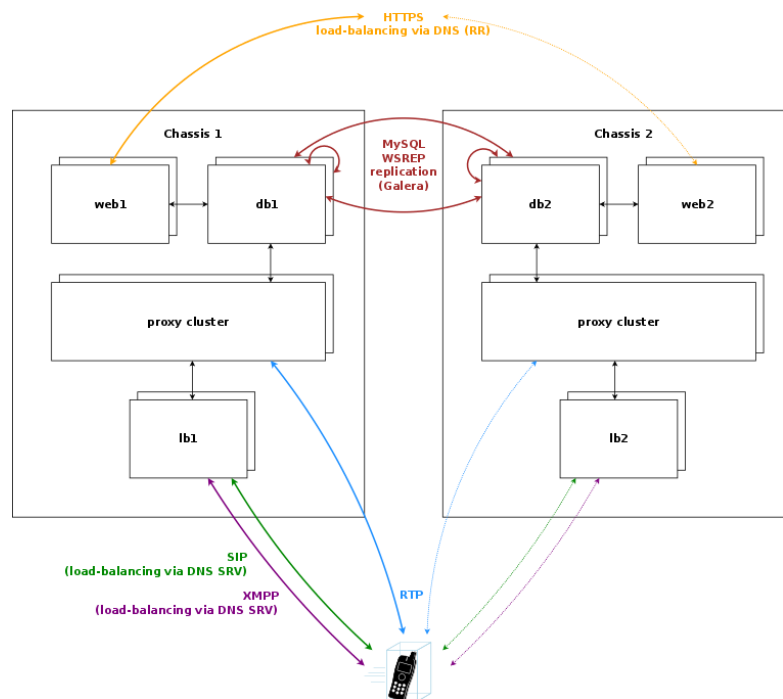


Figure 10: Scaling beyond one chassis

If the sip:carrier is scaled beyond 250.000 subscribers and therefore exceeds one chassis, a second chassis is put into place. This chassis provides another two web servers, two db servers, two load balancers and 8 proxies, doubling the capacity of the system.

### 2.2.3.1 Scaling the DB cluster

The DB cluster is the only node type which requires a notable change on the architecture. Once more than one db pair is deployed, the replication mechanism between db nodes changes from master/master between the nodes of the db1 pair to a synchronous multi-master replication over all db nodes on the system using Galera. This change makes it possible to scale both read and write requests over multiple nodes, while being transparent to all other nodes.

### 2.2.3.2 Scaling the proxy cluster

New proxy nodes replicate via master/slave from the first db node in the chassis as usual. Since the db cluster holds all provisioning information of all subscribers, the proxy nodes join the cluster transparently and will start serving subscribers as soon as all services on a new proxy are reachable from the load balancers.

### 2.2.3.3 Scaling the load balancers

Load balancers are completely stateless, so they start serving subscribers as soon as they are made visible to the subscribers. This could either be done via DNS round-robin, but the better approach is to configure a DNS SRV record, which allows for more fine-grained control like weighting load-balancer pairs and allowing fail-over from one pair to another on the client side.

The load balancers use the Path extension of SIP to make sure during SIP registration that calls targeted to a subscriber are routed via the same load balancer pair which the subscriber used during registration for proper traversal of symmetric NAT at the customer premise.

A SIP or XMPP request reaching a load balancer can be routed to any available proxy in the whole system, or only to proxies belonging to the same chassis as the load balancer, depending on the system configuration.

### 2.2.3.4 Scaling the web servers

New web server pairs are made available to web clients via DNS round-robin. Any pair of web servers can be used to read or write provisioning information via the web interfaces or the API.

## 2.2.4 Architecture for central core and local satellites

---

### Tip

This architecture is not part of the standard deployment and is to be defined in the project plan!

---

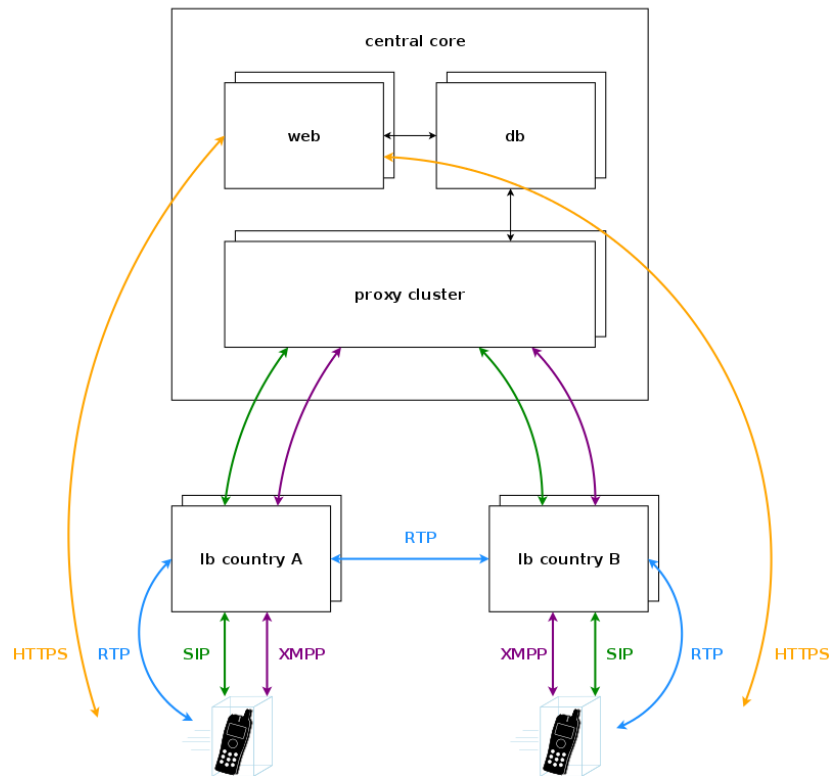


Figure 11: Central core with local breakouts

In case of a geographically distributed system spanning across multiple countries, different regulatory requirements have to be met for signalling and media, especially when it comes to if, where and how subscriber traffic can be intercepted. Countries might have the requirement to intercept traffic in the country, so the signalling and media must be anchored to an element in the country. Also if a media stream stays within a country, it is preferred to keep the media as close to the subscribers as possible to reduce latency, so relaying streams via a central core has to be avoided.

For this scenario, the sip:carrier makes it possible to move the load balancers directly into the countries. DNS settings for subscribers within the country ensure that they will always contact those load balancers, either using separate DNS settings per country for a SIP domain, or using GeoIP mechanisms in DNS to return the closest load balancer based on the location of the subscriber. To anchor media to the countries, the rtpengine instances are moved from the proxies to the load balancers and are controlled via the stateless kamailio instances on the load balancers instead of the kamailio instances on the proxies.

## 3 VoIP Service Administration Concepts

### 3.1 Contacts

A contact contains information such as the name, the postal and email addresses, and others. A contact's main purpose is to identify entities (resellers, customers, peers and subscribers) it is associated with.

A person or an organization may represent a few entities and it is handy to create a corresponding organization's contact beforehand and use it repeatedly when creating new entities. In this case we suggest populating the **External #** field to distinguish between customers associated with the same contact.

Reseller	Contact	Customer	External #
Default	Rylic Longstaff	DTS	0007
		Morning Times	0008
TelephOne	Clare Fenn	Lantern Co	—
	Ike Leonard	City Bank	—

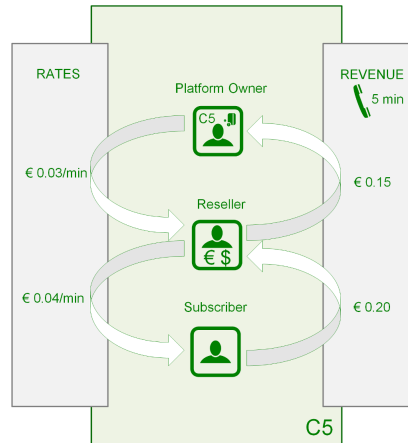
Note that the only required contact field is **email**. For contacts associated with customers, it will be used for sending invoices and notifications such as password reset, new subscriber creation and others. A contact for a subscriber is created automatically but only if you specify an email address for this subscriber. It is mainly used to send notification messages, e.g. in case of a password reset.

### 3.2 Resellers

The reseller model allows you to expand your presence in the market by including virtual operators in the sales chain. A virtual operator can be a company without its own VoIP platform and even without a technical background, but with sales presence in a market. You define such a company as a reseller in the platform: grant limited access to the administrative web interface (the reseller administrator will only see his own customers, domains and billing profiles) and define wholesale rates for this reseller. Then, the reseller is free to operate under its own brand, make up its retail rates, establish the customer base and resell your services to its customers. The reseller's profit is a margin between the wholesale and retail rates.

Let us consider an example:

- You operate in Munich and provide residential and business services.
- A company Cheap Call that has a strong presence in Frankfurt offers to resell your services under its own brand in this city.
- You define wholesale rates for Cheap Call, such as calls to Argentina at €0,03.
- Cheap Call defines its retail price and offers calls to Argentina at €0,04.
- When one of Cheap Call's subscribers makes a 5-minute call to Argentina, this subscriber will be charged €0,20.
- You will get €0,15 revenue and Cheap Call's profit will be €0,20 - €0,15 = €0,05.



A reseller usually uses dedicated IP addresses or SIP domain names to provide services. Also, a reseller can rebrand the self-care web interface for its customers and select languages per SIP domain that allows the reseller to operate even in multiple countries.

### 3.3 SIP Domain

A SIP domain represents an external Internet address where your subscribers register their SIP phones to make calls or send messages. The SIP domain also contains particular default configuration for all the subscribers registered with this SIP domain. A SIP domain can be a regular FQDN (e.g. sip.yourdomain.com) or a NAPTR/SRV record. Using IP addresses for SIP domains in production is **strongly discouraged**.

#### 3.3.1 Additional SIP Domains

You can create as many SIP domains as required to satisfy your networking or marketing requirements, e.g.:

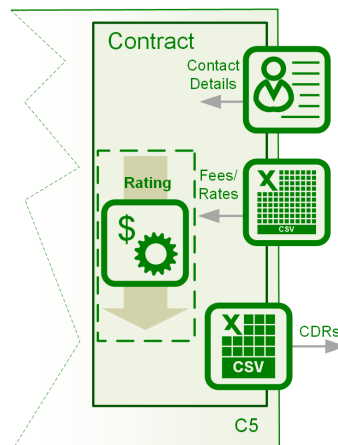
- A dedicated SIP domain is *suggested* per CloudPBX customer.
- A separate SIP domain may be dedicated to every whitelabel reseller.
- Multiple SIP domains may be used to provide services in different countries or regions.
- Multiple SIP domains may be used to brand your own services.

Domain	Purpose
sip.yourdomain.com	Your own domain for retail customers
sip.enterprise.com	Your big customer with Cloud PBX
sip.reseller.com	Your white-label reseller
sip.yourdomain.de	Your domain for providing a new service in another country

### 3.4 Contracts

A contract is a combination of a *contact* and a *billing profile*, hence it represents a business contract for your resellers and peering partners.

Contracts can be created in advance on the *Reseller and Peering Contracts* page, or immediately during creation of a peer or a reseller.



Note that the *customer* entity (described below) is a special type of the contract. A customer entity has an email and an invoice templates in addition to a contact and a billing profile.

### 3.5 Customers

A customer is a physical or legal entity whom you provide the VoIP service with and send invoices to. Here are the main features of a customer:

- Contains the contact and legal information. For example, an address or an email address for invoicing.
- Associated with a billing profile (to define fees per destination) and tracks the balance (used mostly for post-paid customers).
- Contains a certain number of subscribers who actually use the service and whose calls appear in the customer's list of CDRs.
- Provides some default parameters for all its subscribers. For example, voice prompts and call restriction.

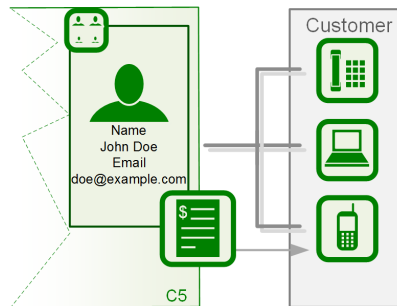
Here are two common examples of the customer model:

#### 3.5.1 Residential and SOHO customers

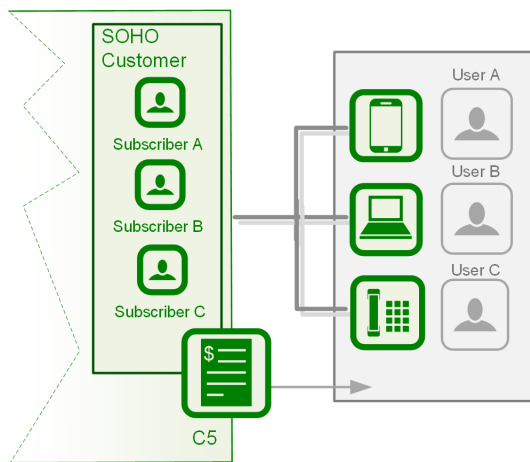
With this service you provide your residential and SOHO customers with one or multiple numbers and offer the service on a post-paid basis.

For a residential customer you usually create one *customer* entity with one *subscriber* under it. A residential customer can register multiple devices with the same number thus having a convenient Viber or Skype-like service: any device can be used to make a

call and all of them will ring simultaneously when there is an incoming call. At the end of the billing period, you send an invoice to the customer.

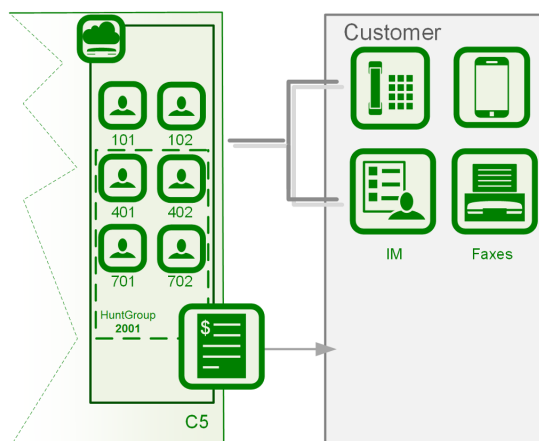


For SOHO customers you usually create multiple subscribers under the same customer and assign every subscriber a dedicated number to allow users make and receive calls. A common invoice will contain calls of all the subscribers.



### 3.5.2 Business customers with the Cloud PBX service

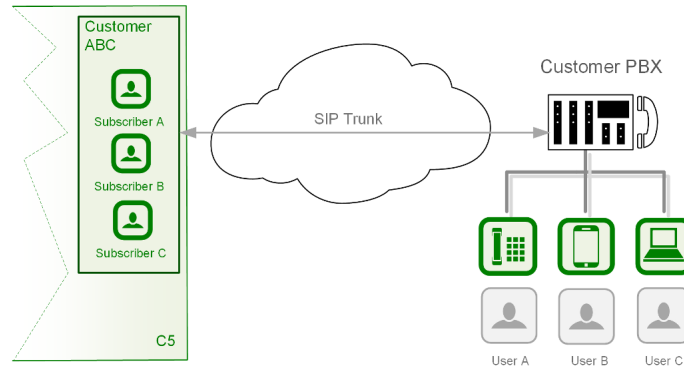
In this case you create a Customer and all the required entities under it to reflect the company's structure: subscribers, extensions, hunt groups, auto-attendant menus, etc.





### 3.5.3 SIP Trunking

If a customer PBX can register itself with C5, you create a regular subscriber for it and configure a standard username/password authentication. Multiple PBX users can then send and receive calls.



Legacy PBX devices that are not capable of passing the *challenge*-based authentication can be authenticated by the IP address. Optionally, every user of such a PBX can be authenticated separately by the FROM header and the IP address. For more details, refer to the [Trusted Sources](#) section.

### 3.5.4 Mobile subscribers

The pre-paid model works perfectly for **mobile application users**. In this case you generally create a single subscriber under a customer.

### 3.5.5 Pre-paid subscribers who use your calling cards

In this case you will most likely create a single subscriber under a customer, although multiple subscribers would work as well. In the latter case, they will share and top-up the common balance. Notice that the *customer* entity itself does not contain any technical configuration for the VoIP service authentication and instead contains other entities called *subscribers*, which do.

## 3.6 Subscribers

Every subscriber represents a SIP line or a SIP trunk. For example, in the residential services a subscriber entity is dedicated to every user. In the SIP trunking scenario, a subscriber can be used to authenticate all VoIP traffic from the remote PBX device.

In the following table logical subscriber types and their purpose are described.

Service	Subscriber Type	Purpose	Features
Residential	Regular subscriber	A regular VoIP service	Requires a DID number to receive calls from outside of your network
Enterprise (CloudPBX)	Pilot subscriber	A base number for the enterprise customer; Lists all extra numbers (aliases)	Configures the rest of customer subscribers in its self-care web interface

Service	Subscriber Type	Purpose	Features
	Extension	Extra numbers (DIDs, "implicit" extensions) for the enterprise customer	Can be dialed in different ways; The number configuration builds on top of the Pilot subscriber
	PBX Group	Forwards incoming calls to multiple extensions	Ringling policy defines in which order the extensions will ring
SIP Trunk	Digest authentication	Dynamically registers a remote IP PBX device	Handles multiple users behind the IP PBX device
	IP authentication	IP authentication of legacy IP PBX devices incapable of registering with the platform	Might require Trusted Subscriber and Trusted Source configuration
Prepaid	Regular subscriber with prepaid billing profile	Authorization of services based on customer balance; Disconnection of calls on "zero balance"	<a href="#">Voucher</a> and <a href="#">cache top-up</a> ; <a href="#">Billing Profile Packages</a>

---

**Tip**

Subscriber **Aliases** can provide Extra DIDs or extension numbers to a subscriber.

---

### 3.7 SIP Peerings

A SIP peering is your interconnection with the external VoIP or PSTN network. Usually, a VoIP service provider has at least a few termination partners to offer its subscribers calls to virtually any landline and mobile destination.

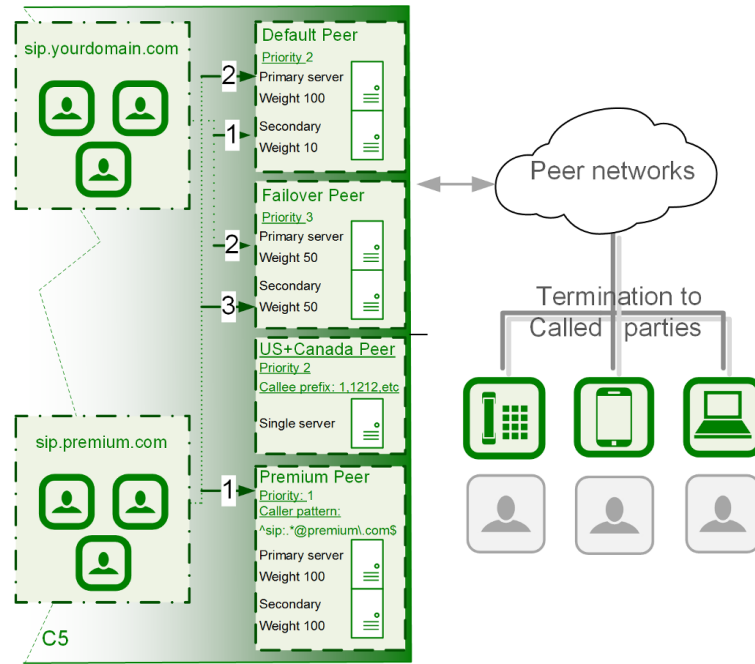
SIP peerings also enable incoming calls to your platform. For example, if you rent a pool of DID numbers from a SIP peer and offer them to your residential and business customers.

An interconnection with your termination partners and DID number providers can include multiple servers and enable both out-bound and inbound calls, hence such a configuration is called a *SIP peering group*. You configure at least one SIP peering group for every partner and the main principle here is that all servers in a group terminate calls to the same set of listed destinations.

Any SIP peering group is associated with a *contract* for reconciliation and billing purposes and includes two main technical configurations:

- Peering Servers Represent connections to/from your SIP peering's network. The parameters include an IP address and/or a hostname of the remote part. For outbound calls, this is the destination address where to send calls to and for inbound calls it is an IP authorization of the remote server.
- Outbound/Inbound Peering Rules Outbound rules define through which SIP peering group a call from a specific subscriber will be sent for termination to a specific destination.

The example below shows four SIP peering groups with different priorities, callee prefixes (actual destinations offered by this SIP peering) and callee / called patterns (fine-tuning which callee request URIs and caller URIs are allowed through this SIP peering group).



The figure shows how calls from premium subscribers can in the first place be routed through a dedicated SIP peering group unavailable to regular subscribers.

See the [Routing Order Selection](#) section for details about call routing.

Inbound rules allow [filtering out incoming INVITE requests](#) arriving from the corresponding SIP peering servers.

## 4 VoIP Service Configuration Scenario

A basic VoIP service configuration is fast, easy and straight-forward. Provided that your network and required DNS records have been preconfigured, the configuration of a VoIP service can be done purely via the administrative web interface. The configuration mainly includes the following steps:

- Reseller creation (optional)
- SIP domain configuration
- Customer creation
- Subscribers provisioning

Let us assume you are using the `1.2.3.4` IP address with an associated `sip.yourdomain.com` domain to provision VoIP services. This allows you to provide an easy-to-remember domain name instead of the IP address as the proxy server. Also, your subscribers' URIs will look like `1234567@sip.yourdomain.com`.

---

### Tip

Using an IP address instead of an associated FQDN (domain name) for a SIP domain is not suggested as it could add extra administrative work if you decide to relocate your servers to another datacenter or just change IP addresses.

---

Go to the *Administrative Web Panel (Admin Panel)* running on `https://<ip>:1443/login/admin` and follow the steps below. The default web panel user and password are *administrator*, if you have not already changed it.

### 4.1 Creating a SIP Domain

A SIP domain is a connection point for your subscribers. The SIP domain also contains specific default configuration for all its subscribers.

---

### Tip

Thoroughly plan your domain names policy in advance and take into account that: 1) the name of a SIP domain cannot be changed after creating it in the administrative web panel; 2) subscribers cannot be moved from one domain to another and must be recreated.

---

To create a SIP domain, follow these steps:

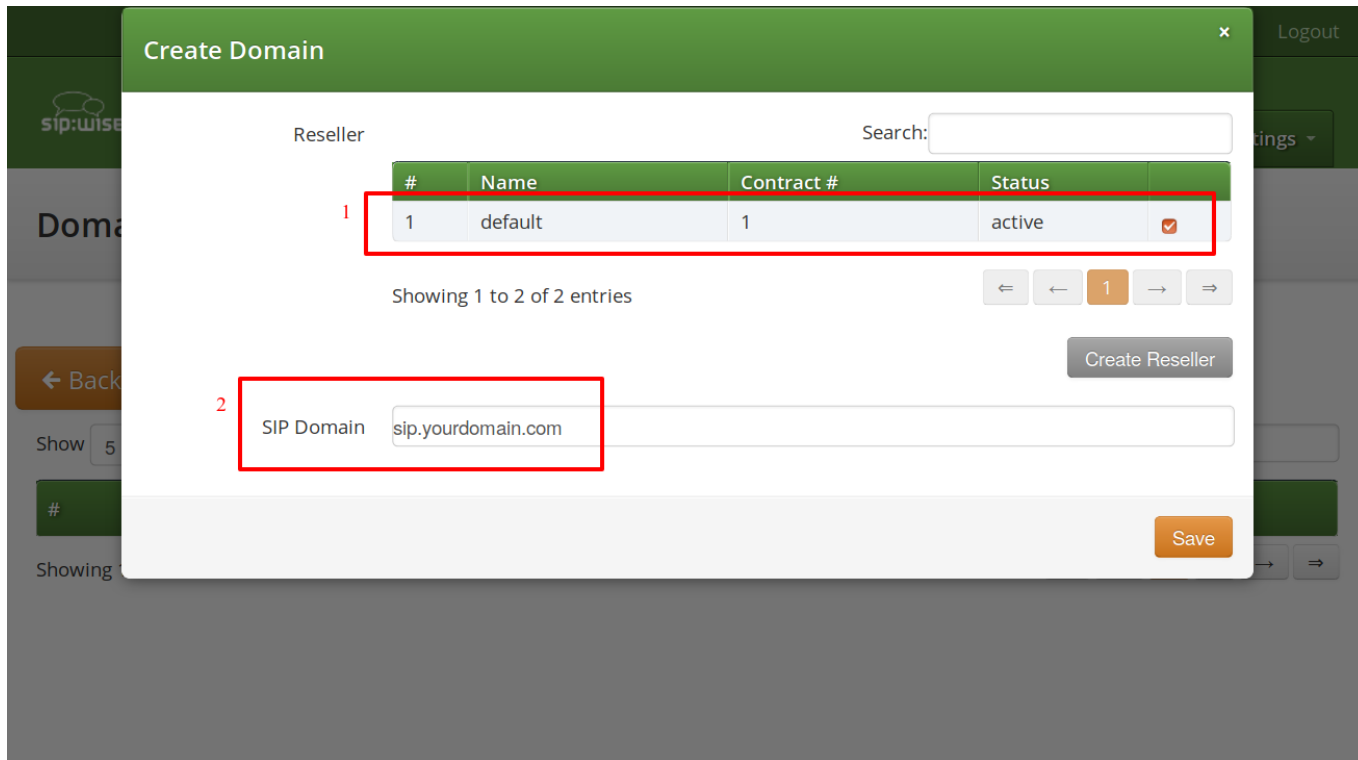
1. Firstly, configure an FQDN on your DNS server for it.

The domain name must point to the physical IP address you are going to use for providing the VoIP service. A good approach is to create an SRV record:

```
SIP via UDP on port 5060
SIP via TCP on port 5060
SIP via TCP/TLS on port 5061
```

2. Create a new SIP domain in the administrative web panel.

Go to the *Domains* page and create a new SIP Domain using the FQDN created above.



Select a *Reseller* who will own the subscribers in this SIP domain. Use the *default* virtual reseller if you provide services directly. Enter your SIP domain name and press *Save*.

3. Adjust the new SIP domain's preferences if necessary.

You can create multiple SIP domains reusing the existing IP address or adding a new one. Extra SIP domains are required e.g. if you would like to host a virtual operator on your platform, create separate domains for providing services in different countries or just offer a new service.

## 4.2 Creating a Customer

A Customer is a special type of contract acting as legal and billing information container for SIP subscribers. A customer can have one or more SIP subscriber entities that represent SIP lines.

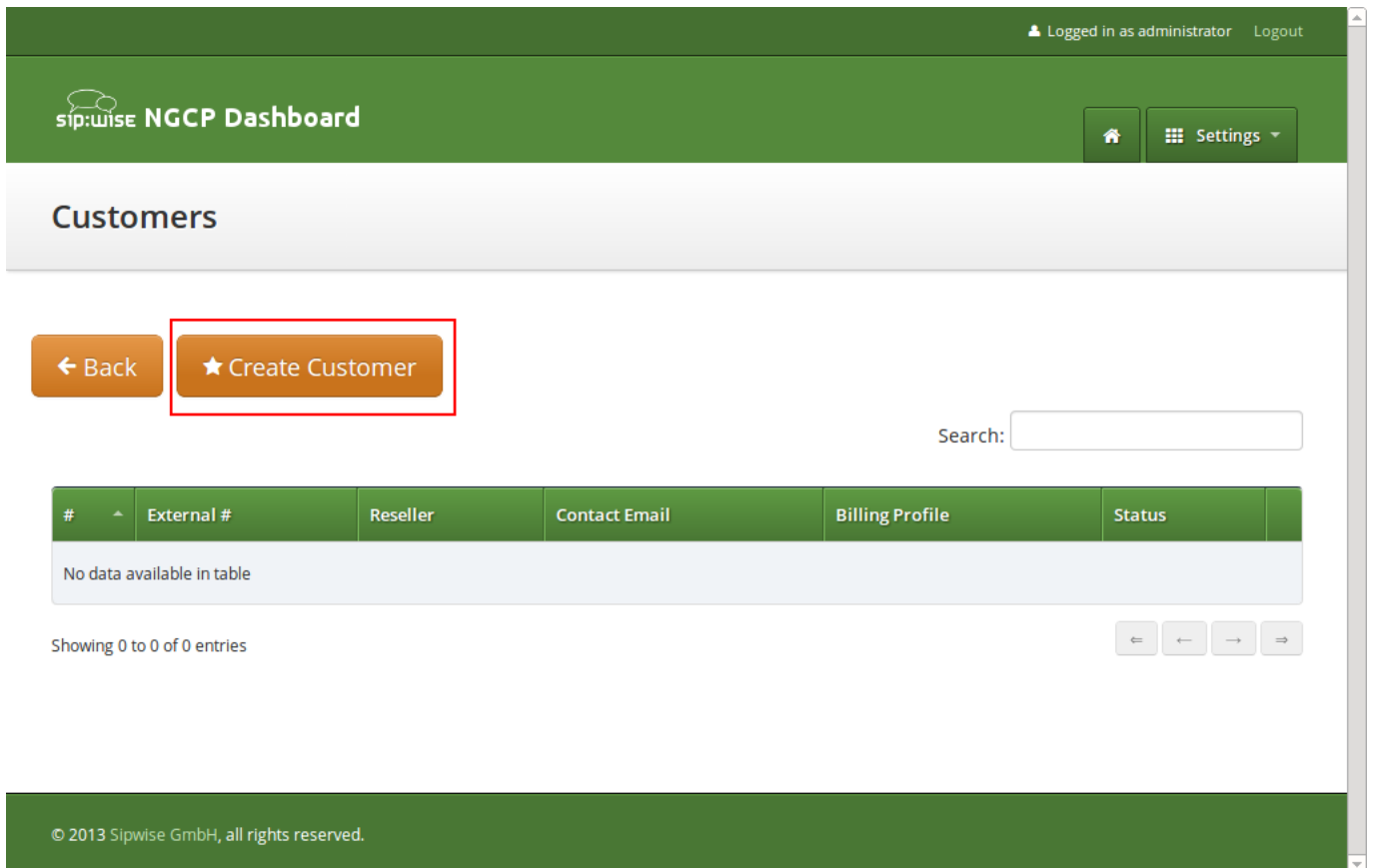
### Tip

For correct billing, notification and invoicing, create a customer with a single SIP subscriber for the residential service (as it normally has only one telephone line) and a customer with multiple SIP subscribers to provide a service to a company with many telephone lines.

To create a Customer, go to *Settings*→*Customers*.

The screenshot shows the sip:wise NGCP Dashboard interface. At the top right, it indicates the user is logged in as an administrator. The main header contains the sip:wise logo and the text 'NGCP Dashboard'. A 'Settings' dropdown menu is open, with 'Customers' highlighted in orange. The dashboard features four main sections: System Status (All services running), Resellers (9 Resellers), Billing (6 Billing Profiles), and a fourth section partially visible. Each section includes a 'Configure' button. The 'Customers' option in the settings menu is the target of the instruction.

Click on *Create Customer*.



The screenshot shows the sip:wise NGCP Dashboard interface. At the top right, it indicates the user is logged in as 'administrator' with a 'Logout' link. The dashboard title is 'sip:wise NGCP Dashboard'. Below the title, there are navigation buttons for 'Home' and 'Settings'. The main section is titled 'Customers'. Below this, there are two buttons: 'Back' and 'Create Customer'. The 'Create Customer' button is highlighted with a red rectangular box. To the right of these buttons is a search input field labeled 'Search:'. Below the search field is a table with the following columns: '#', 'External #', 'Reseller', 'Contact Email', 'Billing Profile', and 'Status'. The table is currently empty, displaying the message 'No data available in table'. Below the table, it shows 'Showing 0 to 0 of 0 entries' and a set of pagination controls.

Each *Customer* has a *Contact*—a container for the personal and legal information that identifies a private or corporate customer.

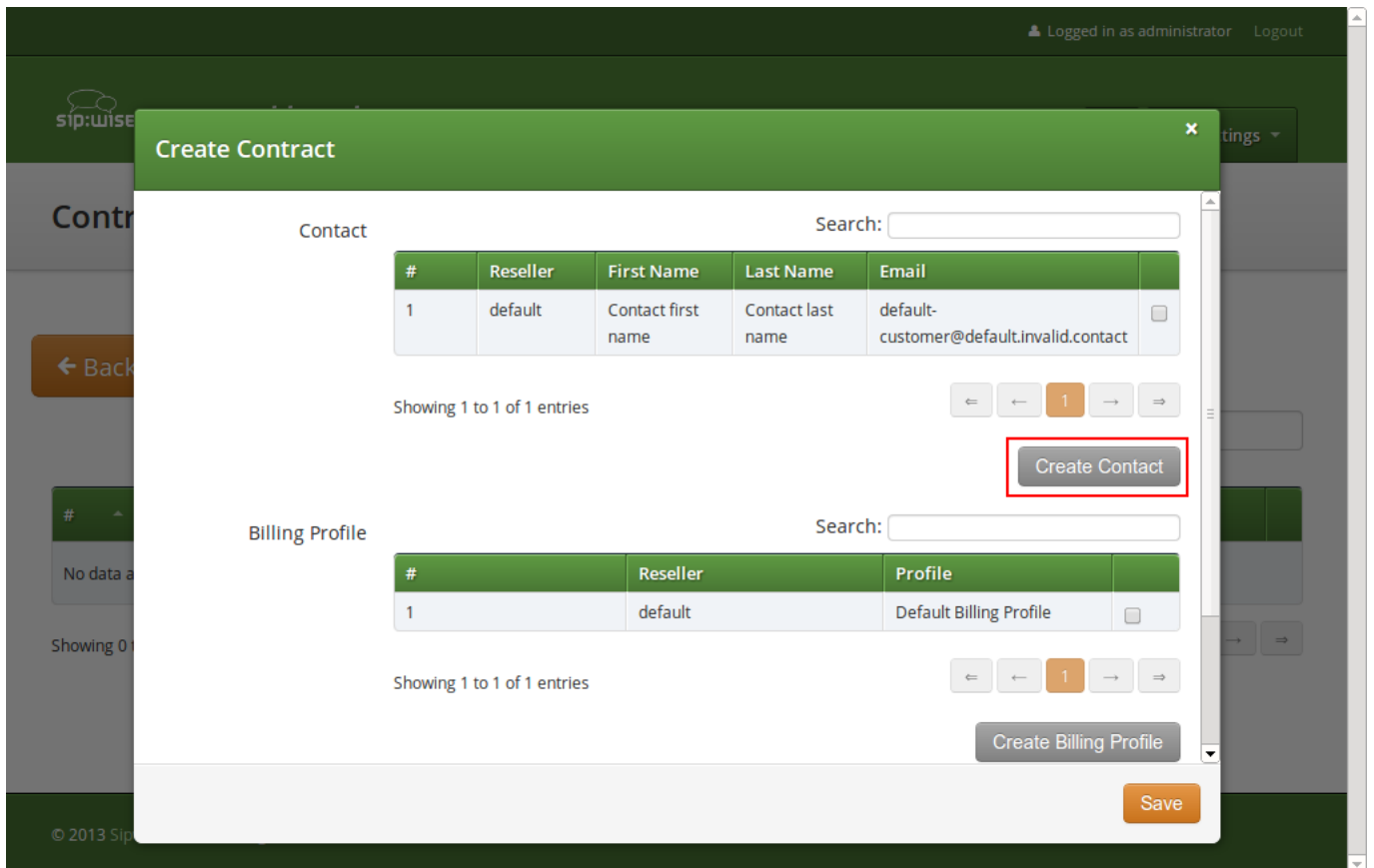
---

**Tip**

Create a dedicated *Contact* for every *Customer* as it contains specific data e.g. name, address and IBAN that identifies this customer.

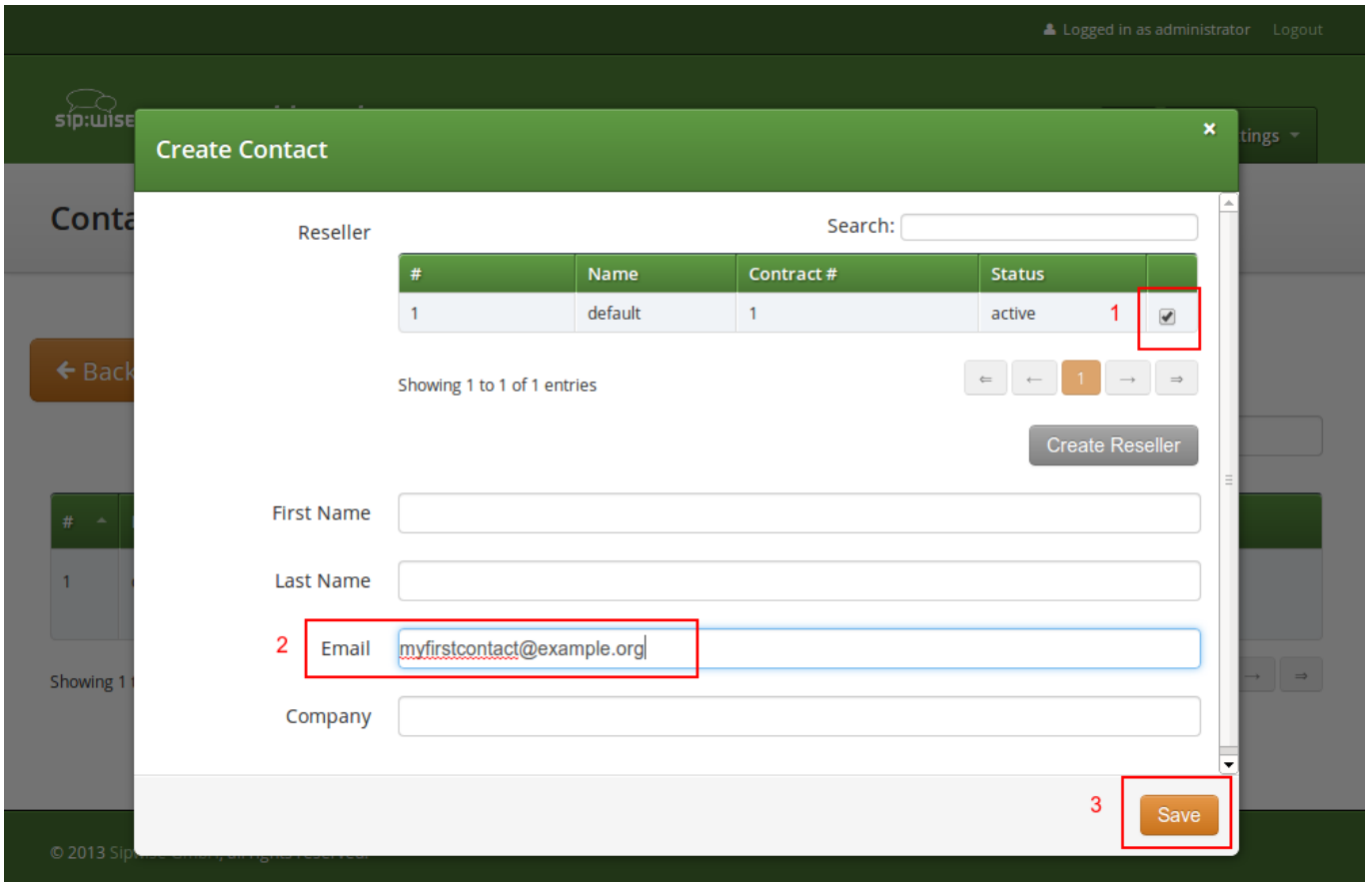
---

Click on *Create Contact* to create a new *Contact*.



Select the required *Reseller* and enter the contact details (at least an *Email* is required), then press *Save*.





You will be redirected back to the *Customer* form. The newly created *Contact* is selected by default now, so only select a *Billing Profile* and press *Save*.

You will now see your first *Customer* in the list. Hover over the customer and click *Details* to make extra configuration if necessary.

Logged in as administrator Logout

sip:wise NGCP Dashboard

Home Settings

## Customers

← Back ★ Create Customer

Contract successfully created

Search:

#	External #	Reseller	Contact Email	Billing Profile	Status	
20		default	myfirstcontact@example.org	Default Billing Profile	active	<a href="#">Edit</a> <a href="#">Terminate</a> <a href="#">Details</a>

Showing 1 to 1 of 1 entries

← 1 →

### 4.3 Creating a Subscriber

In your *Customer* details view, click on the *Subscribers* row, then click *Create Subscriber*.

Logged in as administrator Logout

sip:wise NGCP Dashboard

Home Settings

## Customer Details

Back Edit

- Reseller
- Contact Details
- Billing Profiles
- Subscribers **1**
- ★ Create Subscriber **2**

SIP URI	Primary Number	Registered Devices
---------	----------------	--------------------

Contract Balance

Select a *SIP Domain* created earlier and specify required and optional parameters:

- **Domain:** The domain part of the SIP URI for your subscriber.
- **E164 Number:** This is the telephone number mapped to the subscriber, separated into *Country Code (CC)*, *Area Code (AC)* and *Subscriber Number (SN)*. For the first tests, you can set an imaginary number here and change it later when you get number blocks assigned by your PSTN interconnect partner. So in our example, we'll use *43* as CC, *99* as AC and *1001* as SN to form the imaginary number *+43 99 1001*.

### Tip

This number can actually be used to place calls between local subscribers, even if you don't have any PSTN interconnection. This comes in handy if you use phones instead of soft-clients for your tests. The format in which this number can be dialled, so the subscriber is reached is defined in Section 4.7.

### Important



NGCP allows a single subscriber to have multiple E.164 numbers to be used as aliases for receiving incoming calls. Also, NGCP supports so-called "implicit" extensions. If a subscriber has phone number 012345, but somebody calls 012345100, then NGCP first tries to send the call to number 012345100 (even though the user is registered as 012345). If NGCP then receives the 404 - Not Found response, it falls back to 012345 (the user-part with which the callee is registered).

- **Email:** An email address for sending service-related notifications to.
- **Web Username:** This is the user part of the username the subscriber may use to log into her *Customer Self Care Interface*. The user part will be automatically suffixed by the SIP domain you choose for the **SIP URI**. Usually, the web username is identical to the **SIP URI**, but you may choose a different naming schema.

**Caution**

The web username needs to be unique. The system will return a fault if you try to use the same web username twice.

---

- **Web Password:** This is the password for the subscriber to log into her *Customer Self Care Interface*. It must be at least 6 characters long.
- **SIP Username:** The user part of the SIP URI for your subscriber.
- **SIP Password:** The password of your subscriber to authenticate on the SIP proxy. It must be at least 6 characters long.
- **Status:** You can lock a subscriber here, but for creating one, you will most certainly want to use the *active* status.
- **External ID:** You can provision an arbitrary string here (e.g. an ID of a 3rd party provisioning/billing system).
- **Administrative:** If you have multiple subscribers in one account and set this option for one of them, this subscriber can administrate other subscribers via the *Customer Self Care Interface*.

Domain Search:

#	Reseller	Domain	
113	default	sip.yourdomain.com	<input checked="" type="checkbox"/>

Showing 1 to 1 of 1 entries

Create Domain

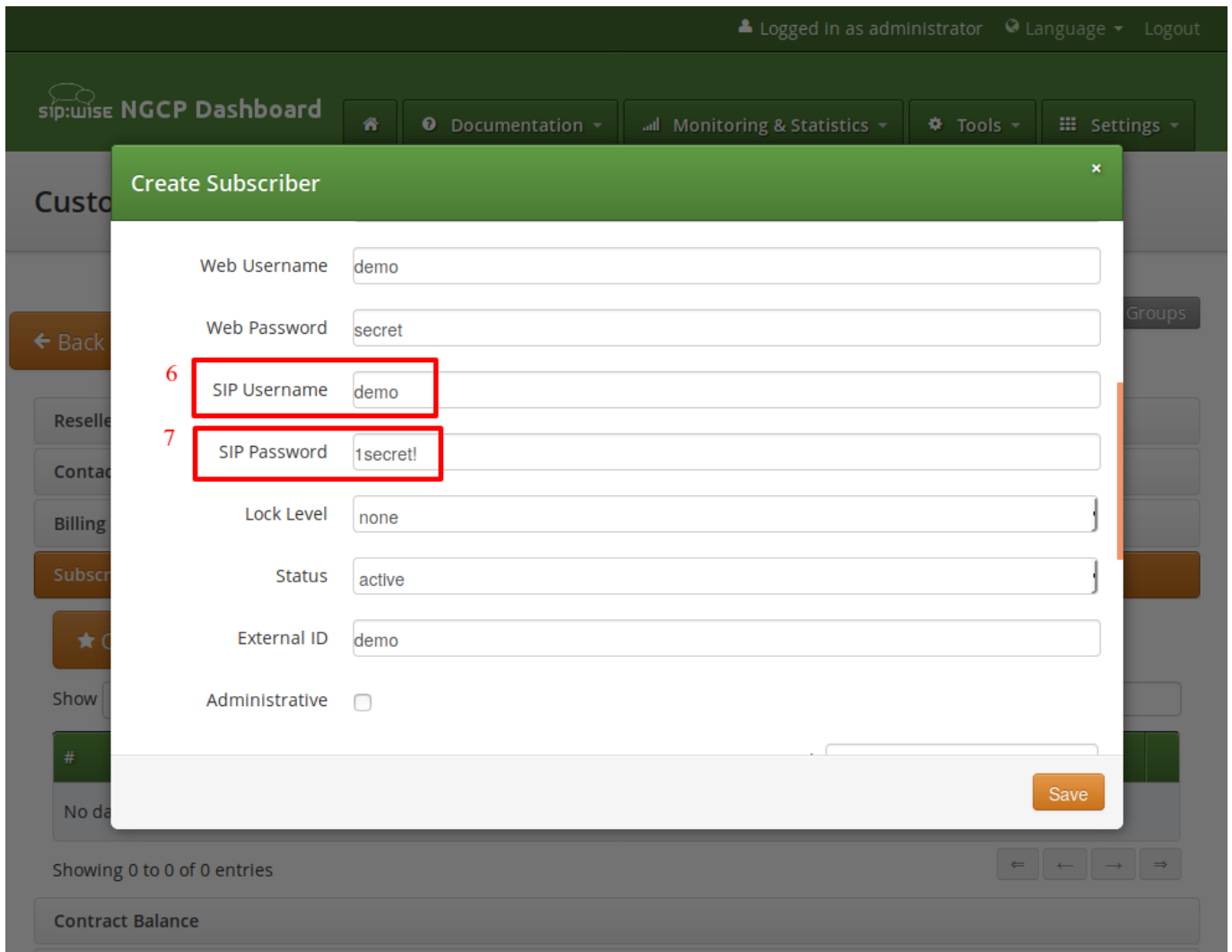
2 E164 Number

3 Email

4 Web Username

5 Web Password

Save



The screenshot shows the 'Create Subscriber' form in the sip:wise NGCP Dashboard. The form is a modal window with a green header. It contains the following fields:

- Web Username: demo
- Web Password: secret
- SIP Username: demo (highlighted with a red box and labeled 6)
- SIP Password: 1secret! (highlighted with a red box and labeled 7)
- Lock Level: none
- Status: active
- External ID: demo
- Administrative:

A 'Save' button is located at the bottom right of the form. The background shows the dashboard navigation menu with options like 'Documentation', 'Monitoring & Statistics', 'Tools', and 'Settings'.

Repeat the creation of *Customers* and *Subscribers* for all your test accounts. You should have at least 3 subscribers to test the functionality of the NGCP.

---

**Tip**

At this point, you're able to register your subscribers to the NGCP and place calls between these subscribers.

---

You should now revise the *Domain* and *Subscriber* Preferences.

#### 4.4 Domain Preferences

The *Domain Preferences* are the default settings for *Subscriber Preferences*, so you should set proper values there if you don't want to configure each subscriber separately. You can later override these settings in the *Subscriber Preferences* if particular subscribers need special settings. To configure your *Domain Preferences*, go to *Settings*→*Domains* and click on the *Preferences* button of the domain you want to configure.

Logged In as administrator Language Logout

sip:wise NGCP Dashboard Documentation Monitoring & Statistics Tools Settings

## Domains

← Back ★ Create Domain

Show 5 entries Search:

#	Reseller	Domain	
113	default	sip.yourdomain.com	Delete Preferences

Showing 1 to 1 of 1 entries

The most important settings are in the *Number Manipulations* group.

Here you can configure the following:

- for incoming calls - which SIP message headers to take numbers from
- for outgoing calls - where in the SIP messages to put certain numbers to
- for both - how these numbers are normalized to E164 format and vice versa

To assign a *Rewrite Rule Set* to a *Domain*, create a set first as described in Section 4.7, then assign it to the domain by editing the *rewrite\_rule\_set* preference.

## Domain "sip.yourdomain.com" - Preferences

← Back

Call Blockings

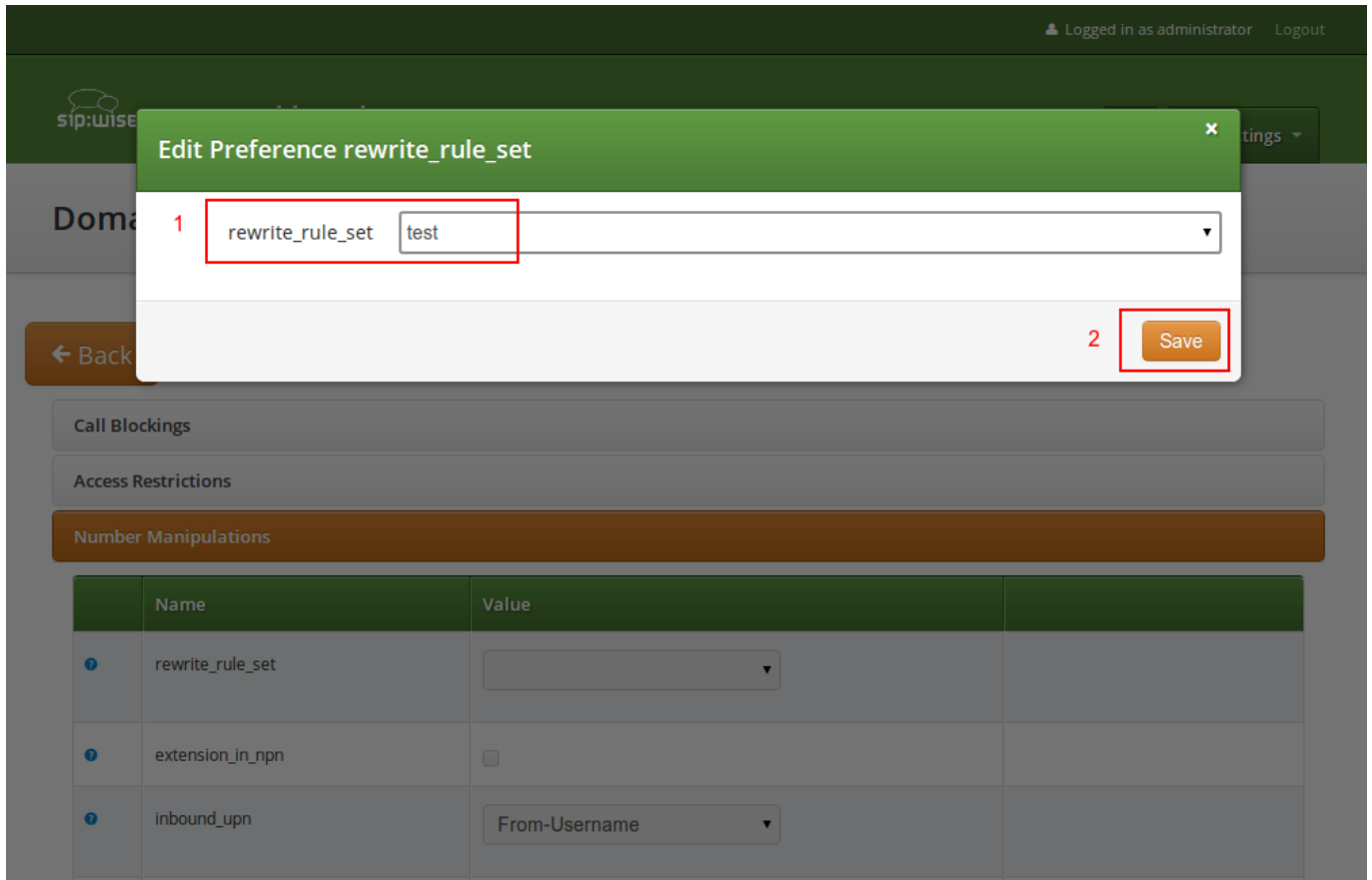
Access Restrictions

1 Number Manipulations

	Name	Value	
?	rewrite_rule_set	<input type="text" value=""/>	2 <input type="button" value="Edit"/>
?	extension_in_npn	<input type="checkbox"/>	
?	inbound_upn	<input type="text" value="From-Username"/>	
?	outbound_from_user	<input type="text" value="User-Provided-Number"/>	
?	outbound_from_display	<input type="text" value="None"/>	

Select the *Rewrite Rule Set* and press *Save*.





Then, select the field you want the *User Provided Number* to be taken from for inbound INVITE messages. Usually the *From-Username* should be fine, but you can also take it from the *Display-Name* of the From-Header, and other options are available as well.

## 4.5 Subscriber Preferences

You can override the *Domain Preferences* on a subscriber basis as well. Also, there are *Subscriber Preferences* which don't have a default value in the *Domain Preferences*.

To configure your *Subscriber*, go to *Settings*→*Subscribers* and click *Details* on the row of your subscriber. There, click on the *Preferences* button on top.

You want to look into the *Number Manipulations* and *Access Restrictions* options in particular, which control what is used as user-provided and network-provided calling numbers.

- For outgoing calls, you may define multiple numbers or patterns to control what a subscriber is allowed to send as user-provided calling numbers using the *allowed\_clis* preference.
- If *allowed\_clis* does not match the number sent by the subscriber, then the number configured in *cli* (the network-provided number) preference will be used as user-provided calling number instead.
- You can override any user-provided number coming from the subscriber using the *user\_cli* preference.

**Note**

Subscribers preference *allowed\_clis* will be synchronized with subscribers primary number and aliases if *oss-bss→provisioning→auto\_allow\_cli* is set to **1** in */etc/ngcp-config/config.yml*.

**Note**

Subscribers preference *cli* will be synchronized with subscribers primary number if *ossbss→provisioning→auto\_sync\_cli* is set to **yes** in */etc/ngcp-config/config.yml*.

## 4.6 Creating Peerings

If you want to terminate calls at or allow calls from 3<sup>rd</sup> party systems (e.g. PSTN gateways, SIP trunks), you need to create SIP peerings for that. To do so, go to *Settings→Peerings*. There you can add peering groups, and for each peering group add peering servers and rules controlling which calls are routed over these groups. Every peering group needs a peering contract for correct interconnection billing.

### 4.6.1 Creating Peering Groups

Click on *Create Peering Group* to create a new group.

In order to create a group, you must select a peering contract. You will most likely want to create one contract per peering group.

The screenshot shows the 'Create SIP Peering Groups' modal dialog in the Sipwise web interface. The dialog is titled 'Create SIP Peering Groups' and has a green header. It contains a table for selecting a contract, a search bar, and form fields for Name, Priority, and Description. A 'Create Contract' button is highlighted with a red box.

Contract Search:

#	Status	Billing Profile
No data available in table		

Showing 0 to 0 of 0 entries

Name

Priority

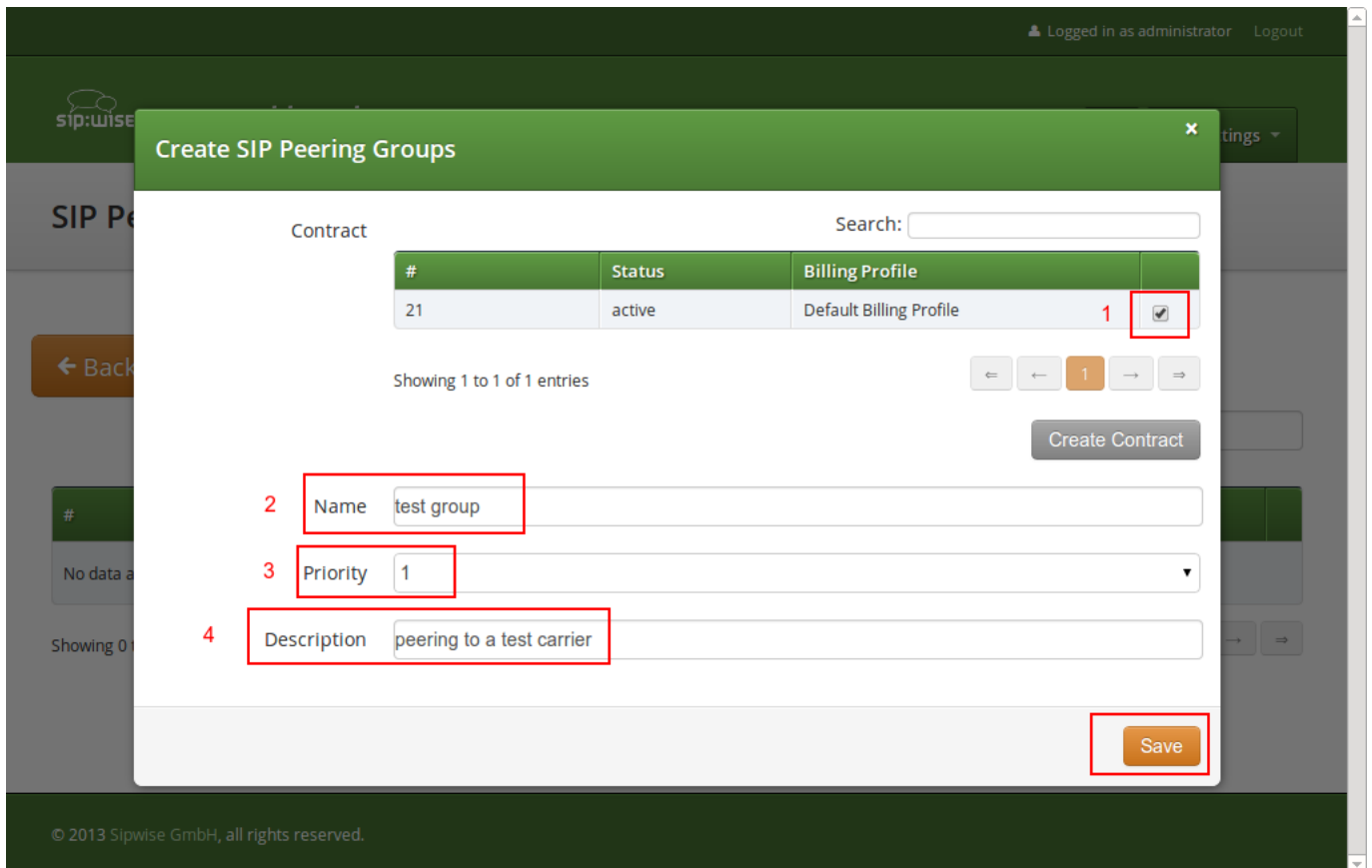
Description

Click on *Create Contract* create a *Contact*, then select a *Billing Profile*.

The screenshot shows the 'Create Contract' modal window. It contains two tables for selection. The 'Contact' table has two rows, with the first row (ID 1) selected. The 'Billing Profile' table has one row (ID 1) selected. A 'Create Contact' button is visible between the tables, and a 'Save' button is at the bottom right. Red boxes and numbers highlight the selected contact ID (1.1), the 'Create Contact' button (labeled 'or 1.2'), the selected billing profile ID (2), and the 'Save' button (labeled '3').

Click *Save* on the *Contacts* form, and you will get redirected back to the form for creating the actual *Peering Group*. Put a name, priority and description there, for example:

- **Peering Contract:** select the id of the contract created before
- **Name:** test group
- **Priority:** 1
- **Description:** peering to a test carrier



The *Priority* option defines which *Peering Group* to favor (Priority 1 gives the highest precedence) if two peering groups have peering rules matching an outbound call. *Peering Rules* are described below.

Then click *Save* to create the group.

#### 4.6.2 Creating Peering Servers

In the group created before, you need to add peering servers to route calls to and receive calls from. To do so, click on *Details* on the row of your new group in your peering group list.

To add your first *Peering Server*, click on the *Create Peering Server* button.

**Peering Servers**

← Back **★ Create Peering Server**

Show 5 entries Search:

#	Name	IP Address	Hostname	Port	Protocol	Weight	Via Route Set	Enabled
No data available in table								

Showing 0 to 0 of 0 entries

**Outbound Peering Rules**

ANY of the rules must match to choose the peering group for outbound calls.

**★ Create Outbound Peering Rule**

Show 5 entries Search:

#	Callee Prefix	Callee Pattern	Caller Pattern	Description	Enabled
No data available in table					

Showing 0 to 0 of 0 entries

**Inbound Peering Rules**

ALL of the rules must match to choose the peering group for inbound calls.

**★ Create Inbound Peering Rule**

Show 5 entries Search:

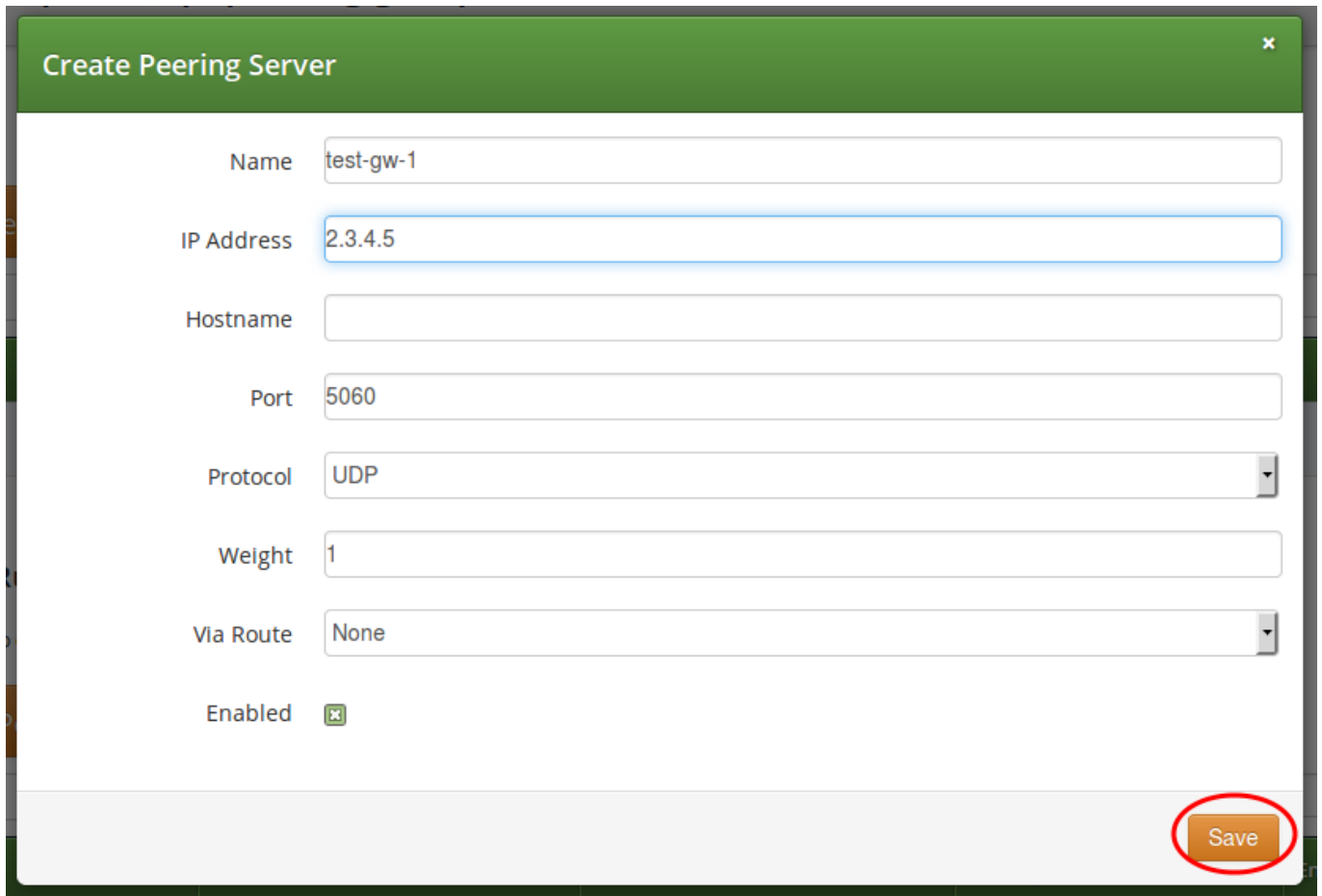
Priority	#	Field	Pattern	Reject Code	Reject Reason	Enabled
No data available in table						

Showing 0 to 0 of 0 entries

Figure 12: Create Peering Server

In this example, we will create a peering server with IP 2.3.4.5 and port 5060:

- **Name:** test-gw-1
- **IP Address:** 2.3.4.5
- **Hostname:** leave empty
- **Port:** 5060
- **Protocol:** UDP
- **Weight:** 1
- **Via Route:** None



**Create Peering Server**

Name

IP Address

Hostname

Port

Protocol

Weight

Via Route

Enabled

**Save**

Figure 13: Peering Server Properties

Click **Save** to create the peering server.

---

**Tip**

The *hostname* field for a peering server is optional. Usually, the IP address of the peer is used as the **domain** part of the Request URI. Fill in this field if a peer requires a particular hostname instead of the IP address. The IP address must always be given though as the request will always be sent to the specified IP address, no matter what you put into the *hostname* field.

---

---

**Tip**

If you want to add a peering server with an IPv6 address, enter the address without surrounding square brackets into the *IP Address* column, e.g. `::1`.

---

You can force an additional hop (e.g. via an external SBC) towards the peering server by using the *Via Route* option. The available options you can select there are defined in `/etc/ngcp-config/config.yml`, where you can add an array of SIP URIs in `kamailio→lb→external_sbc` like this:

```
kamailio:
  lb:
```

```
external_sbc:
- sip:192.168.0.1:5060
- sip:192.168.0.2:5060
```

Execute `ngcpcfg apply "added external sbc gateways"`, then edit your peering server and select the hop from the *Via Route* selection.

Once a peering server has been created, this server can already send calls to the system.

#### 4.6.2.1 Outbound Peering Rules



#### Important

To be able to send outbound calls towards the servers in the *Peering Group*, you also need to define *Outbound Peering Rules*. They specify which source and destination numbers are going to be terminated over this group. To create a rule, click the *Create Outbound Peering Rule* button.

**Peering Servers**

← Back    ★ Create Peering Server

Peering server successfully created

Show 5 entries    Search:

#	Name	IP Address	Hostname	Port	Protocol	Weight	Via Route Set	Enabled
29	test-gw-1	2.3.4.5		5060	1	1		1

Showing 1 to 1 of 1 entries

**Outbound Peering Rules**

ANY of the rules must match to choose the peering group for outbound calls.

★ Create Outbound Peering Rule

Show 5 entries    Search:

#	Callee Prefix	Callee Pattern	Caller Pattern	Description	Enabled
No data available in table					

Showing 0 to 0 of 0 entries

**Inbound Peering Rules**

ALL of the rules must match to choose the peering group for inbound calls.

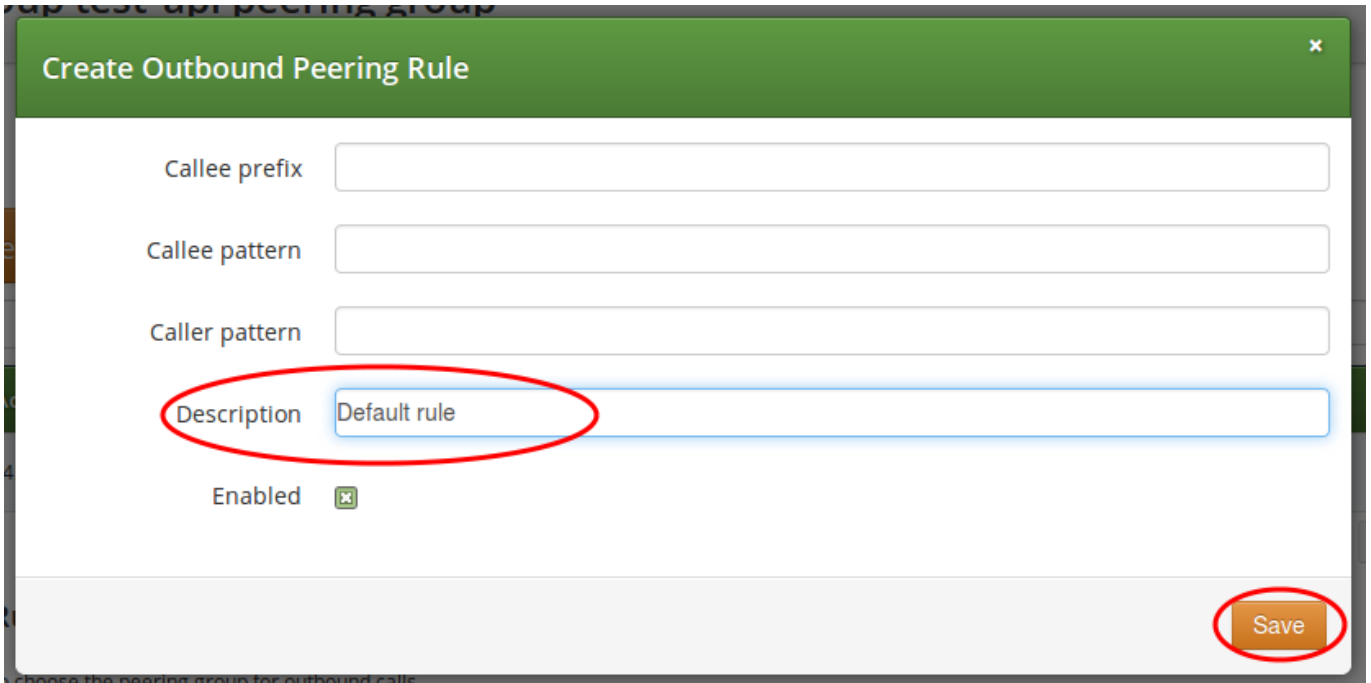
★ Create Inbound Peering Rule

Figure 14: Create Outbound Peering Rule

Since the previously created peering group will be the only one in our example, we have to add a default rule to route *all* calls via

this group. To do so, create a new peering rule with the following values:

- **Callee Prefix:** leave empty
- **Callee Pattern:** leave empty
- **Caller Pattern:** leave empty
- **Description:** Default Rule



The screenshot shows a dialog box titled "Create Outbound Peering Rule". It contains four input fields: "Callee prefix", "Callee pattern", "Caller pattern", and "Description". The "Description" field is highlighted with a red circle and contains the text "Default rule". Below the input fields is a checkbox labeled "Enabled" which is checked. At the bottom right of the dialog is a "Save" button, also circled in red.

Figure 15: Outbound Peering Rule Properties

Then click *Save* to add the rule to your group.

---

**Tip**

In contrast to the callee/caller pattern, the callee prefix has a regular alphanumeric string and can not contain any regular expression.

---

---

**Tip**

If you set the caller or callee rules to refine what is routed via this peer, enter all phone numbers in full E.164 format, that is `<cc><ac><sn>`.

---

---

**Tip**

The *Caller Pattern* field covers the whole URI including the subscriber domain, so you can only allow certain domains over this peer by putting for example `@example\.com` into this field.

---



### 4.6.2.2 Inbound Peering Rules

Starting from *mr5.0* release, Sipwise NGCP supports filtering SIP INVITE requests sent by SIP peers. The system administrator may define one or more matching rules for SIP URIs that are present in the headers of SIP INVITE requests, and select which SIP header (or part of the header) must match the pattern declared in the rule.

If the incoming SIP INVITE message has the proper headers, NGCP will accept and further process the request. If the message does not match the rule it will be rejected.



**Caution**

An incoming SIP INVITE message must match **all the inbound peering rules** so that NGCP does not reject the request.

In order to **create an inbound peering rule** you have to select a peering group, press *Details* and then press *Create Inbound Peering Rule* button.

#### Peering Servers

← Back
★ Create Peering Server

Show 5 entries
Search:

#	Name	IP Address	Hostname	Port	Protocol	Weight	Via Route Set	Enabled
29	test-gw-1	2.3.4.5		5060	1	1		1

Showing 1 to 1 of 1 entries

---

#### Outbound Peering Rules

ANY of the rules must match to choose the peering group for outbound calls.

★ Create Outbound Peering Rule

Show 5 entries
Search:

#	Callee Prefix	Callee Pattern	Caller Pattern	Description	Enabled
1				Default rule	1

Showing 1 to 1 of 1 entries

---

#### Inbound Peering Rules

ALL of the rules must match to choose the peering group for inbound calls.

★ Create Inbound Peering Rule

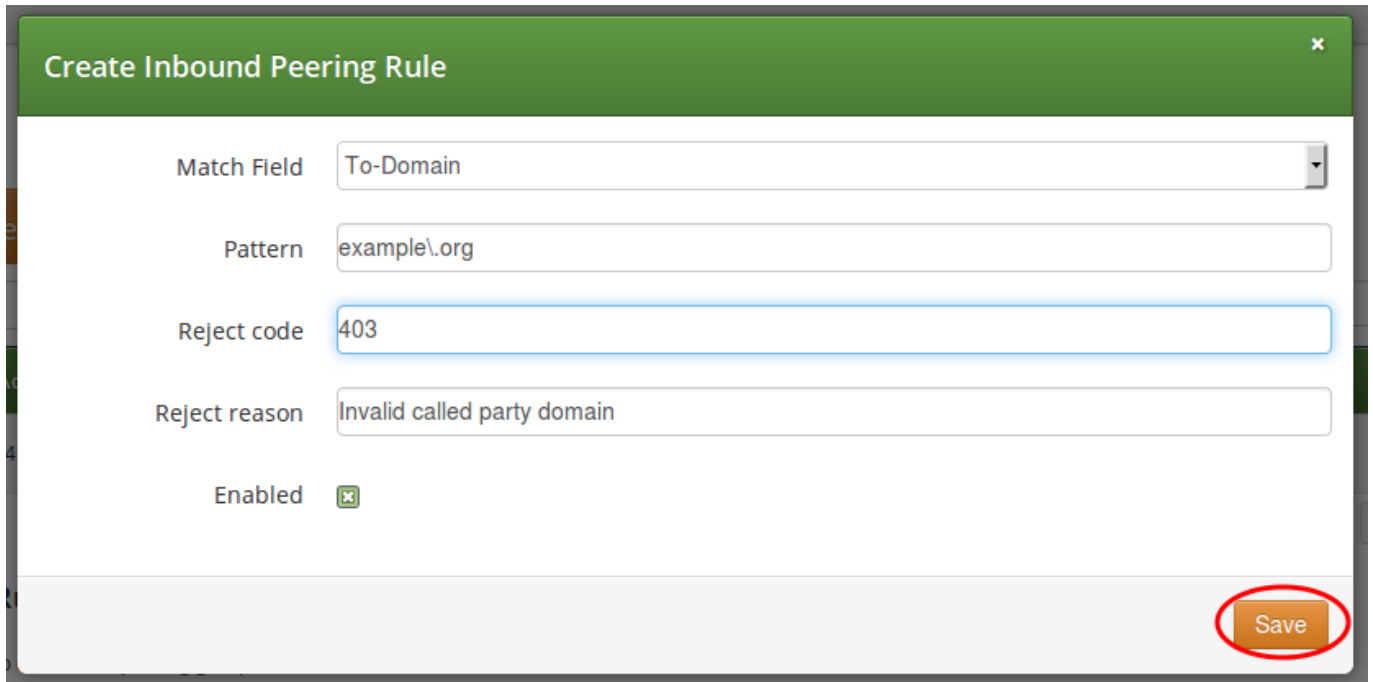
Show 5 entries
Search:

Priority	#	Field	Pattern	Reject Code	Reject Reason	Enabled
No data available in table						

Showing 0 to 0 of 0 entries

Figure 16: Create Inbound Peering Rule

An inbound peering rule has the following **properties**:



The screenshot shows a dialog box titled "Create Inbound Peering Rule". It contains the following fields and values:

- Match Field: To-Domain
- Pattern: example\.org
- Reject code: 403
- Reject reason: Invalid called party domain
- Enabled:

A "Save" button is located at the bottom right of the dialog, highlighted with a red circle.

Figure 17: Inbound Peering Rule Properties

- `Match Field`: select which header and which part of that header in a SIP INVITE message will be checked for matching the pattern
- `Pattern`: a POSIX regular expression that defines the accepted value of a header; example: `^sip:.*@example\.org$` —this will match a SIP URI that contains "example.org" in the domain part
- `Reject code`: optional; a SIP status code that will be sent as a response to an INVITE request that does not match the pattern; example: 403
- `Reject reason`: optional; an arbitrary text that will be included in the SIP response sent with the *reject code*
- `Enabled`: a flag to enable / disable the particular inbound peering rule

---

#### Note

Both of the properties `Reject code` and `Reject reason` must be left empty if a peering server (i.e. a specific IP address) is part of more peering groups. Such a configuration is useful when an incoming SIP INVITE request needs to be treated differently in the affected peering groups, based on its content, and that's why if the INVITE message only partly matches an inbound peering rule it should not simply be rejected.

---

When all settings for a peering group are done the details of the group look like:

### Peering Servers

[← Back](#)
[★ Create Peering Server](#)

Show 5 entries Search:

#	Name	IP Address	Hostname	Port	Protocol	Weight	Via Route Set	Enabled
29	test-gw-1	2.3.4.5		5060	1	1		1

Showing 1 to 1 of 1 entries ← ← 1 → →

### Outbound Peering Rules

ANY of the rules must match to choose the peering group for outbound calls.

[★ Create Outbound Peering Rule](#)

Show 5 entries Search:

#	Callee Prefix	Callee Pattern	Caller Pattern	Description	Enabled
1				Default rule	1

Showing 1 to 1 of 1 entries ← ← 1 → →

### Inbound Peering Rules

ALL of the rules must match to choose the peering group for inbound calls.

[★ Create Inbound Peering Rule](#)

Show 5 entries Search:

Priority	#	Field	Pattern	Reject Code	Reject Reason	Enabled
50	1	to_domain	example\org	403	Invalid called party domain	1

Showing 1 to 1 of 1 entries ← ← 1 → →

Figure 18: Peering Servers Overview

#### 4.6.2.3 Routing Order Selection

The selection of peering groups and peering servers for outgoing calls is done in the following way:

- All peering groups that meet the following criteria configured in the outbound peering rule are added to the list of routes for a particular call:
  - Callee's username matches *callee prefix*
  - Callee's URI matches *callee pattern*
  - Caller's URI matches *caller pattern*
- When all matching peering groups are selected, they are ordered by *callee prefix* according to the **longest match basis** (sometimes referred to as the **longest pattern match** or **maximum pattern length match**). One or more peering group with longest *callee prefix* match will be given first positions on the list of routes.

3. Peering groups with the same *callee prefix* length are further ordered by **Priority**. Peering group(s) with the higher priorities will occupy higher positions.



#### Important

Priority **1** gives the *highest* precedence to the corresponding peering group. Hence, a lower priority value will put the peering group higher in the list of routes (compared to other peering groups with the same *callee prefix* length).

Priority can be selected from **1** (highest) to **9** (lowest).

4. All peering servers in the peering group with the highest priority (e.g. priority **1**) are tried one-by-one starting from the highest server weight. Peering groups with lower priorities or with shorter *callee prefix* will be used only for fail-over.

The **weight** of the peering servers in the selected peering group will influence the order in which the servers within the group will be tried for routing the outbound call. The weight of a server can be set in the range from **1** to **127**.



#### Important

Opposite to the peering group priority, a peering server with a higher weight value has a *higher* precedence, but the server weight rather sets a probability than a strict order. E.g. although a peering server with weight **127** has the highest chance to be the first in the list of routes, another server with a lower weight (e.g. **100**) sometimes will be selected first.

In order to find out this probability knowing the weights of peering servers, use the following script:

```
#!/usr/bin/php
<?php

// This script can be used to find out actual probabilities
// that correspond to a list of peering weights.

if ($argc < 2) {
    echo "Usage: lcr_weight_test.php <list of weights (integers 1-254)>\n";
    exit;
}

$iters = 10000;

$rand = array();
for ($i = 1; $i <= $iters; $i++) {
    $elem = array();
    for ($j = 1; $j < $argc; $j++) {
        $elem["$j"] = $argv[$j] * (rand() >> 8);
    }
    $rand[] = $elem;
}

$sorted = array();
```

```
foreach ($rands as $rand) {
    asort($rand);
    $sorted[] = $rand;
}

$countss = array();
for ($j = 1; $j < $argc; $j++) {
    $countss["$j"] = 0;
}

foreach ($sorted as $rand) {
    end($rand);
    $countss[key($rand)]++;
}

for ($j = 1; $j < $argc; $j++) {
    echo "Peer with weight " . $argv[$j] . " has probability " . $countss["$j"]/$iters . "\n";
}
?>
```

Let us say you have 2 peering servers, one with weight 1 and another with weight 2. At the end—running the script as below—you will have the following traffic distribution:

```
# lcr_weight_test.php 1 2

Peer with weight 1 has probability 0.2522
Peer with weight 2 has probability 0.7478
```

If a peering server replies with SIP codes 408, 500 or 503, or if a peering server doesn't respond at all, the next peering server in the current peering group is tried as a fallback. All the servers within the group are tried one after another until the call succeeds. If no more servers are left in the current peering group, the next group which matches the outbound peering rules is used.

---

**Note**

The Sipwise NGCP may use a slightly different approach in selecting the appropriate peering server if the *peer probing* feature is enabled. See the details in Section 5.10 of the handbook.

---

## 4.6.3 Authenticating and Registering against Peering Servers

### 4.6.3.1 Proxy-Authentication for outbound calls

If a peering server requires the sip:carrier to authenticate for outbound calls (by sending a 407 as response to an INVITE), then you have to configure the authentication details in the *Preferences* view of your peer host.

### Peering Servers

← Back
★ Create Peering Server

Show  entries Search:

#	Name	IP Address	Hostname	Port	Protocol	Weight	Via Route Set	Enabled	
29	test-gw-1	2.3.4.5		5060	1	1		1	<span>Edit</span> <span>Delete</span> <span>Preferences</span>

Showing 1 to 1 of 1 entries ← ← 1 → →

### Outbound Peering Rules

ANY of the rules must match to choose the peering group for outbound calls.

★ Create Outbound Peering Rule

Show  entries Search:

#	^	Callee Prefix	Callee Pattern	Caller Pattern	Description	Enabled	
1					Default rule	1	

Showing 1 to 1 of 1 entries ← ← 1 → →

### Inbound Peering Rules

ALL of the rules must match to choose the peering group for inbound calls.

★ Create Inbound Peering Rule

Figure 19: Select Peering Server Preferences

To configure this setting, open the *Remote Authentication* tab and edit the following three preferences:

- **peer\_auth\_user:** <username for peer auth>
- **peer\_auth\_pass:** <password for peer auth>
- **peer\_auth\_realm:** <domain for peer auth>

[← Back](#)




Preference peer\_auth\_realm successfully updated.

Access Restrictions

Number Manipulations

NAT and Media Flow Control

Remote Authentication

	Name	Value	
	peer_auth_user 1	peeruser1	
	peer_auth_pass 2	peerpass1	
	peer_auth_realm 3	testpeering.com	
	peer_auth_register	<input type="checkbox"/>	
	find_subscriber_by_uuid	<input type="checkbox"/>	

Session Timers

### Important



If you do NOT authenticate against a peer host, then the caller CLI is put into the From and P-Asserted-Identity headers, e.g. "+4312345" <sip:+4312345@your-domain.com>. If you DO authenticate, then the From header is "+4312345" <sip:your\_peer\_auth\_user@your\_peer\_auth\_realm> (the CLI is in the Display field, the peer\_auth\_user in the From username and the peer\_auth\_realm in the From domain), and the P-Asserted-Identity header is as usual like <sip:+4312345@your-domain.com>. So for presenting the correct CLI in *CLIP no screening* scenarios, your peering provider needs to extract the correct user either from the From Display-Name or from the P-Asserted-Identity URI-User.

### Tip

You will notice that these three preferences are also shown in the *Subscriber Preferences* for each subscriber. There you can override the authentication details for all peer host if needed, e.g. if every user authenticates with his own separate credentials at your peering provider.

### Tip

If **peer\_auth\_realm** is set, the system may overwrite the Request-URI with the peer\_auth\_realm value of the peer when sending the call to that peer or peer\_auth\_realm value of the subscriber when sending a call to the subscriber. Since this is rarely a desired behavior, it is disabled by default starting with NGCP release 3.2. If you need the replacement, you should set `set_ruri_to_peer_auth_realm: 'yes'` in `/etc/ngcp-config/config.yml`.

### 4.6.3.2 Registering at a Peering Server

Unfortunately, the credentials configured above are not yet automatically used to register the sip:carrier at your peer hosts. There is however an easy manual way to do so, until this is addressed.

Configure your peering servers with the corresponding credentials in `/etc/ngcp-config/templates/etc/ngcp-sems/etc/reg_agent.conf.tt2`, then execute `ngcpcfg apply "added upstream credentials"`.

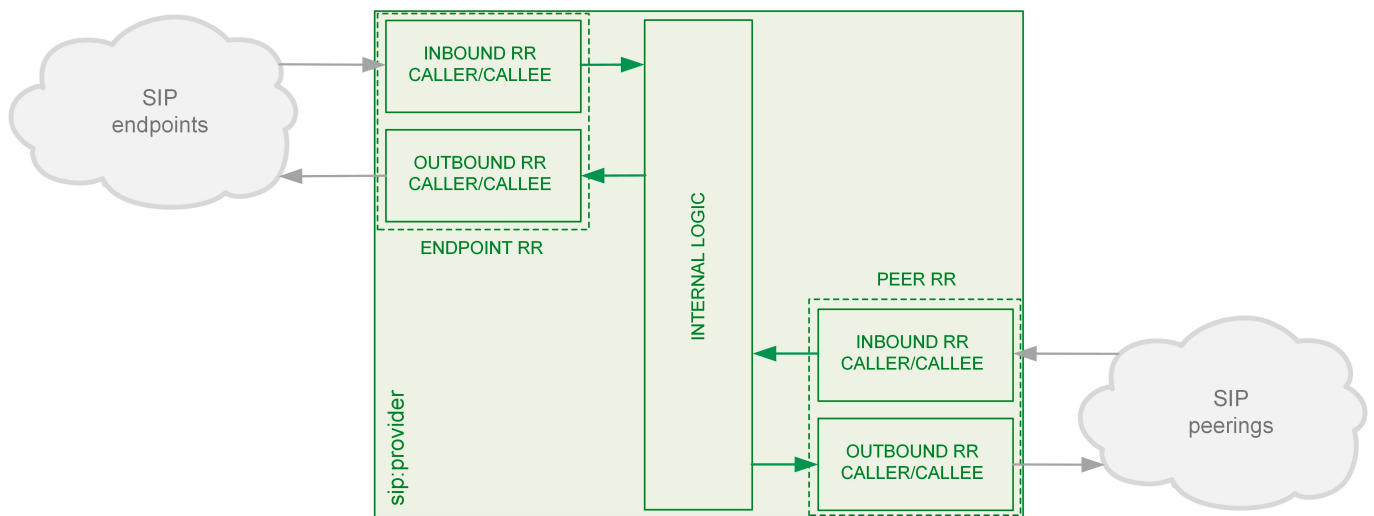


#### Important

Be aware that this will force SEMS to restart, which will drop all calls.

## 4.7 Configuring Rewrite Rule Sets

On the NGCP, every phone number is treated in E.164 format `<country code><area code><subscriber number>`. Rewrite Rule Sets is a flexible tool to translate the caller and callee numbers to the proper format before the routing lookup and after the routing lookup separately. The created Rewrite Rule Sets can be assigned to the domains, subscribers and peers as a preference. Here below you can see how the Rewrite Rules are used by the system:



As from the image above, following the arrows, you will have an idea about which type of Rewrite Rules are applied during a call. In general:

- Call from local subscriber A to local subscriber B: Inbound RR from local Domain/Subscriber A and Outbound Rewrite Rules from local Domain/Subscriber B.
- Call from local subscriber A to the peer: Inbound RR from local Domain/Subscriber A and Outbound Rewrite Rules from the peer.
- Call from peer to local subscriber B: Inbound RR from the Peer and Outbound Rewrite Rules from local Domain/Subscriber B.

You would normally begin with creating a Rewrite Rule Set for your SIP domains. This is used to control what an end user can dial



for outbound calls, and what is displayed as the calling party on inbound calls. The subscribers within a domain inherit Rewrite Rule Sets of that domain, unless this is overridden by a subscriber Rewrite Rule Set preference.

You can use several special variables in the Rewrite Rules, below you can find a list of them. Some examples of how to use them are also provided in the following sections:

- `${caller_cc}` : This is the value taken from the subscriber's preference CC value under Number Manipulation
- `${caller_ac}` : This is the value taken from the subscriber's preference AC value under Number Manipulation
- `${caller_emergency_cli}` : This is the value taken from the subscriber's preference emergency\_cli value under Number Manipulation
- `${caller_emergency_prefix}` : This is the value taken from the subscriber's preference emergency\_prefix value under Number Manipulation
- `${caller_emergency_suffix}` : This is the value taken from the subscriber's preference emergency\_suffix value under Number Manipulation
- `${caller_cloud_pbx_base_cli}` : This is the value taken from the *Primary Number* field from section *Details* → *Master Data* of the *Pilot Subscriber* for a particular PBX customer.

To create a new Rewrite Rule Set, go to *Settings* → *Rewrite Rule Sets*. There you can create a Set identified by a name. This name is later shown in your peer-, domain- and user-preferences where you can select the rule set you want to use.

Logged in as administrator Logout

sip:wise NGCP Dashboard

Settings

## Rewrite Rule Sets

← Back **★ Create Rewrite Rule Set**

Search:

#	Reseller	Name	Description
1	default	defaultdom	Default Domain

Showing 1 to 1 of 1 entries

© 2013 Sipwise GmbH, all rights reserved.

Click *Create Rewrite Rule Set* and fill in the form accordingly.

Logged in as administrator Logout

### Create Rewrite Rule Sets

Reseller Search:

#	Name	Contract #	Status	
1	default	1	active	1 <input checked="" type="checkbox"/>

Showing 1 to 1 of 1 entries

Create Reseller

Name  2

Description  3

4

© 2013 Sipwise GmbH, all rights reserved.

Press the *Save* button to create the set.

To view the *Rewrite Rules* within a set, hover over the row and click the *Rules* button.

Logged in as administrator Logout

sip:wise NGCP Dashboard

Home Settings

## Rewrite Rule Sets

Back Create Rewrite Rule Set

Rewrite rule set successfully created

Search:

#	Reseller	Name	Description	
1	default	defaultdom	Default Domain	
2	default	domain-dialplan	Dialplan for Domains	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Rules</a>

Showing 1 to 2 of 2 entries

The rules are ordered by *Caller* and *Callee* as well as direction *Inbound* and *Outbound*.

### Tip

In Europe, the following formats are widely accepted:  $+<cc><ac><sn>$ ,  $00<cc><ac><sn>$  and  $0<ac><sn>$ . Also, some countries allow the areacode-internal calls where only subscriber number is dialed to reach another number in the same area. Within this section, we will use these formats to show how to use rewrite rules to normalize and denormalize number formats.

#### 4.7.1 Inbound Rewrite Rules for Caller

These rules are used to normalize user-provided numbers (e.g. passed in *From Display Name* or *P-Preferred-Identity* headers) into E.164 format. In our example, we'll normalize the three different formats mentioned above into E.164 format.

To create the following rules, click on the *Create Rewrite Rule* for each of them and fill them with the values provided below.

STRIP LEADING 00 OR +

- Match Pattern:  $^(00|\+)([1-9][0-9]+)\$$
- Replacement Pattern:  $\2$
- Description: International to E.164
- Direction: Inbound

- Field: Caller

REPLACE 0 BY CALLER'S COUNTRY CODE:

- Match Pattern: `^0([1-9][0-9]+)$`
- Replacement Pattern: `${caller_cc}\1`
- Description: National to E.164
- Direction: Inbound
- Field: Caller

NORMALIZE LOCAL CALLS:

- Match Pattern: `^([1-9][0-9]+)$`
- Replacement Pattern: `${caller_cc}${caller_ac}\1`
- Description: Local to E.164
- Direction: Inbound
- Field: Caller

The screenshot shows the 'Create Rule' dialog box in the Sipwise interface. The dialog is a white box with a green header and a close button. It contains several fields: 'Match pattern' with the value '^([00|+)([1-9][0-9]+)\$' and a red box around it labeled '1'; 'Replacement Pattern' with the value '\2' and a red box around it labeled '2'; 'Description' with the value 'International to E.164' and a red box around it labeled '3'; 'Direction' with the value 'Inbound' and a red box around it labeled '4'; 'Field' with the value 'Caller' and a red box around it labeled '5'; and a 'Save' button at the bottom right with a red box around it labeled '6'. The background shows a blurred interface with a 'Back' button and a list of call directions.

Normalization for national and local calls is possible with special variables `${caller_cc}` and `${caller_ac}` that can be used in Replacement Pattern and are substituted by the country and area code accordingly during the call routing.



**Important**

These variables are only being filled in when a call originates from a subscriber (because only then the cc/ac information is known by the system), so you can not use them when a calls comes from a SIP peer (the variables will be just empty in this case).

**Tip**

When routing a call, the rewrite processing is stopped after the first match of a rule, starting from top to bottom. If you have two rules (e.g. a generic one and a more specific one), where both of them would match some numbers, reorder them with the up/down arrows into the appropriate position.

## Rewrite Rules for domain-dialplan

← Back

★ Create Rewrite Rule

Rewrite rule successfully created

Inbound Rewrite Rules for Caller

	Match Pattern	Replacement Pattern	Description
1	↑ ↓ <input type="checkbox"/> <input type="checkbox"/>	^(00 \+)([1-9][0-9]+)\$	\2 International to E.164
	↑ ↓ <input type="checkbox"/> <input type="checkbox"/> 2	^0([1-9][0-9]+)\$	\${caller_cc}\1 National to E.164
	↑ ↓ <input type="checkbox"/> <input type="checkbox"/>	^[1-9][0-9]+)\$	\${caller_cc}\${caller_ac}\1 Local to E.164

Inbound Rewrite Rules for Callee

Outbound Rewrite Rules for Caller

Outbound Rewrite Rules for Callee

### 4.7.2 Inbound Rewrite Rules for Callee

These rules are used to rewrite the number the end user dials to place a call to a standard format for routing lookup. In our example, we again allow the three different formats mentioned above and again normalize them to E.164, so we put in the same rules as for the caller.

STRIP LEADING 00 OR +

- Match Pattern: `^(00|\+)([1-9][0-9]+)$`
- Replacement Pattern: `\2`

- **Description:** International to E.164
- **Direction:** Inbound
- **Field:** Callee

REPLACE 0 BY CALLER'S COUNTRY CODE:

- **Match Pattern:** `^0([1-9][0-9]+)$`
- **Replacement Pattern:** `${caller_cc}\1`
- **Description:** National to E.164
- **Direction:** Inbound
- **Field:** Callee

NORMALIZE AREACODE-INTERNAL CALLS:

- **Match Pattern:** `^([1-9][0-9]+)$`
- **Replacement Pattern:** `${caller_cc}${caller_ac}\1`
- **Description:** Local to E.164
- **Direction:** Inbound
- **Field:** Callee

---

### Tip

Our provided rules will only match if the caller dials a numeric number. If he dials an alphanumeric SIP URI, none of our rules will match and no rewriting will be done. You can however define rules for that as well. For example, you could allow your end users to dial `support` and rewrite that to your support hotline using the match pattern `^support$` and the replace pattern `43800999000` or whatever your support hotline number is.

---

### 4.7.3 Outbound Rewrite Rules for Caller

These rules are used to rewrite the calling party number for a call to an end user. For example, if you want the device of your end user to show `0<ac><sn>` if a national number calls this user, and `00<cc><ac><sn>` if an international number calls, put the following rules there.

REPLACE AUSTRIAN COUNTRY CODE 43 BY 0

- **Match Pattern:** `^43([1-9][0-9]+)$`
- **Replacement Pattern:** `0\1`
- **Description:** E.164 to Austria National

- **Direction:** Outbound
- **Field:** Caller

#### PREFIX 00 FOR INTERNATIONAL CALLER

- **Match Pattern:** `^ ([1-9] [0-9]+) $`
- **Replacement Pattern:** `00\1`
- **Description:** E.164 to International
- **Direction:** Outbound
- **Field:** Caller

---

#### Tip

Note that both of the rules would match a number starting with 43, so reorder the national rule to be above the international one (if it's not already the case).

---

#### 4.7.4 Outbound Rewrite Rules for Callee

These rules are used to rewrite the called party number immediately before sending out the call on the network. This gives you an extra flexibility by controlling the way request appears on a wire, when your SBC or other device expects the called party number to have a particular tech-prefix. It can be used on calls to end users too if you want to do some processing in intermediate SIP device, e.g. apply legal intercept selectively to some subscribers.

#### PREFIX SIPSP# FOR ALL CALLS

- **Match Pattern:** `^ ([0-9]+) $`
- **Replacement Pattern:** `sipsp#\1`
- **Description:** Intercept this call
- **Direction:** Outbound
- **Field:** Callee

#### 4.7.5 Emergency Number Handling

There are 2 ways to handle calls from local subscribers to emergency numbers in NGCP:

- *Simple* emergency number handling: inbound rewrite rules append an emergency tag to the called number, this will be recognised by NGCP's call routing logic and the call is routed directly to a peer. Please read the next section for details of simple emergency number handling.
- An emergency *number mapping* is applied: a dedicated emergency number mapping database is consulted in order to obtain the most appropriate routing number of emergency services. This logic ensures that the caller will contact the geographically closest emergency service. Please visit the [Emergency Mapping](#) Section 5.5 section of the handbook for more details.

#### **4.7.5.1 Simple Emergency Number Handling Overview**

The overview of emergency call processing is as follows:



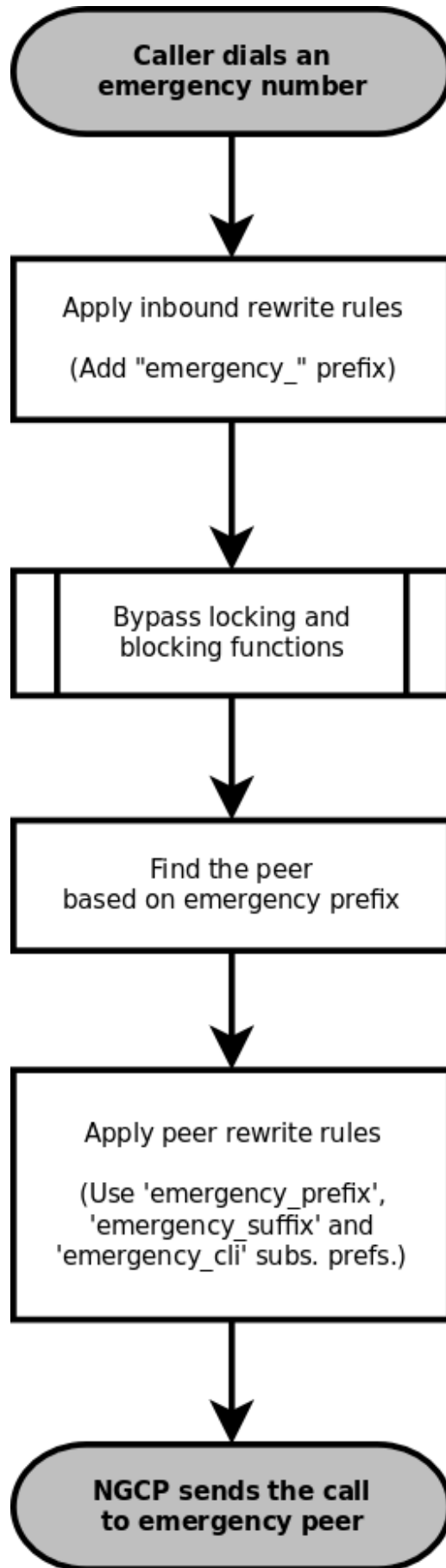


Figure 20: Simple Emergency Call Handling

Configuring Emergency Numbers is also done via Rewrite Rules.

#### 4.7.5.2 Tagging Inbound Emergency Calls

For Emergency Calls from a subscriber to the platform, you need to define an *Inbound Rewrite Rule For Callee*, which adds a prefix `emergency_` to the number (and can rewrite the number completely as well at the same time). If the proxy detects a call to a SIP URI starting with `emergency_`, it will enter a special routing logic bypassing various checks which might make a normal call fail (e.g. due to locked or blocked numbers, insufficient credits or exceeding the max. amount of parallel calls).

##### TAG AN EMERGENCY CALL

- Match Pattern: `^(911|112)$`
- Replacement Pattern: `emergency_\1`
- Description: Tag Emergency Numbers
- Direction: Inbound
- Field: Callee

To route an Emergency Call to a Peer, you can select a specific peering group by adding a peering rule with a *callee prefix* set to `emergency_` to a peering group.

#### 4.7.5.3 Normalize Emergency Calls for Peers

In order to normalize the emergency number to a valid format accepted by the peer, you need to assign an *Outbound Rewrite Rule For Callee*, which strips off the `emergency_` prefix. You can also use the variables `${caller_emergency_cli}`, `${caller_emergency_prefix}` and `${caller_emergency_suffix}` as well as `${caller_ac}` and `${caller_cc}`, which are all configurable per subscriber to rewrite the number into a valid format.

##### NORMALIZE EMERGENCY CALL FOR PEER

- Match Pattern: `^emergency_(.+)$`
- Replacement Pattern: `${caller_emergency_prefix}${caller_ac}\1`
- Description: Normalize Emergency Numbers
- Direction: Outbound
- Field: Callee

#### 4.7.6 Assigning Rewrite Rule Sets to Domains and Subscribers

Once you have finished to define your Rewrite Rule Sets, you need to assign them. For sets to be used for subscribers, you can assign them to their corresponding domain, which then acts as default set for all subscribers. To do so, go to *Settings*→*Domains* and click *Preferences* on the domain you want the set to assign to. Click on *Edit* and select the Rewrite Rule Set created before.

Call Blockings

Access Restrictions 1

Number Manipulations

Name	Value
rewrite_rule_set	defaultdom
extension_in_npn	<input type="checkbox"/>
inbound_upn	From-Username
outbound_from_user	User-Provided-Number

You can do the same in the *Preferences* of your subscribers to override the rule on a subscriber basis. That way, you can finely control down to an individual user the dial-plan to be used. Go to *Settings*→*Subscribers*, click the *Details* button on the subscriber you want to edit, then click the *Preferences* button.

#### 4.7.7 Creating Dialplans for Peering Servers

For each peering server, you can use one of the Rewrite Rule Sets that was created previously as explained in Section 4.7 (keep in mind that special variables `${caller_ac}` and `${caller_cc}` can not be used when the call comes from a peer). To do so, click on the name of the peering server, look for the preference called *Rewrite Rule Sets*.

If your peering servers don't send numbers in E.164 format `<cc><ac><sn>`, you need to create *Inbound Rewrite Rules* for each peering server to normalize the numbers for caller and callee to this format, e.g. by stripping leading + or put them from national into E.164 format.

Likewise, if your peering servers don't accept this format, you need to create *Outbound Rewrite Rules* for each of them, for example to append a + to the numbers.

#### 4.7.8 Call Routing Verification

The sip:carrier provides a utility that helps with the verification of call routing among local subscribers and peers. It is called *Call Routing Verification* and employs rewrite rules and peer selection rules, in order to process calling and called numbers or SIP users and find the appropriate peer for the destination.

The *Call Routing Verification* utility performs only basic number processing and does not invoke the full number manipulation logic applied on real calls. The goal is to enable testing of rewrite rules, rather than validate the complete number processing.

- What is considered during the test:
  - subscriber preferences: `cli` and `allowed_clis`
  - domain / subscriber / peer rewrite rules
- What is not taken into account during the test:
  - other subscriber or peer preferences
  - LNP (Local Number Portability) lookup on called numbers; LNP rewrite rules

You can access the utility following the path on Admin web interface: *Tools* → *Call Routing Verification*.

### **Expected input data**

- `Caller number/uri`: 2 formats are accepted in this field:
  - A simple **phone number** in international (00431 . . , +431 . . ) or E.164 (431 . . ) format.
  - A SIP **URI** in `username@domain` format (without adding "sip:" at the beginning).
- `Callee number/uri`: The same applies as for `Caller number/uri`.
- `Caller Type`: Select `Subscriber` or `Peer`, depending on the source of the call.
- `Caller Subscriber` or `Caller Peer`: Optionally, you can select the subscriber or peer explicitly. Without the explicit selection, however, the *Call Routing Verification* tool is able to find the caller in the database, based on the provided number / URI.
- `Caller RWR Override`, `Callee RWR Override`, `Callee Peer Override`: The caller / callee rewrite rules and peer selection rules defined in domain, subscriber and peer preferences are used for call processing by default. But you can also override them by explicitly selecting another rewrite or peer selection rule.

### **Examples**

1. Using only phone numbers and explicit subscriber selection

- Input Data:

### Call Routing Verification

[← Back](#) [Expand Groups](#)

Caller number/url:   
Callee number/url:

Caller Type:  Subscriber  
 Peer

Caller Subscriber Search:

#	Username	Domain	UUID	Number	
295	43993002	10.15.18.227	51e32173-c8a9-44f1-af30-a1ed431eb2bf		<input checked="" type="checkbox"/>
297	43993003	10.15.18.227	6feb9ea-21c0-4f55-8828-80d546b8998f		<input type="checkbox"/>
299	43993004	10.15.18.227	3543a26e-861b-459f-a348-a8ef3e1e9eab		<input type="checkbox"/>
301	43993005	10.15.18.227	355773d2-1c08-475c-8858-eaf75bb58c73		<input type="checkbox"/>

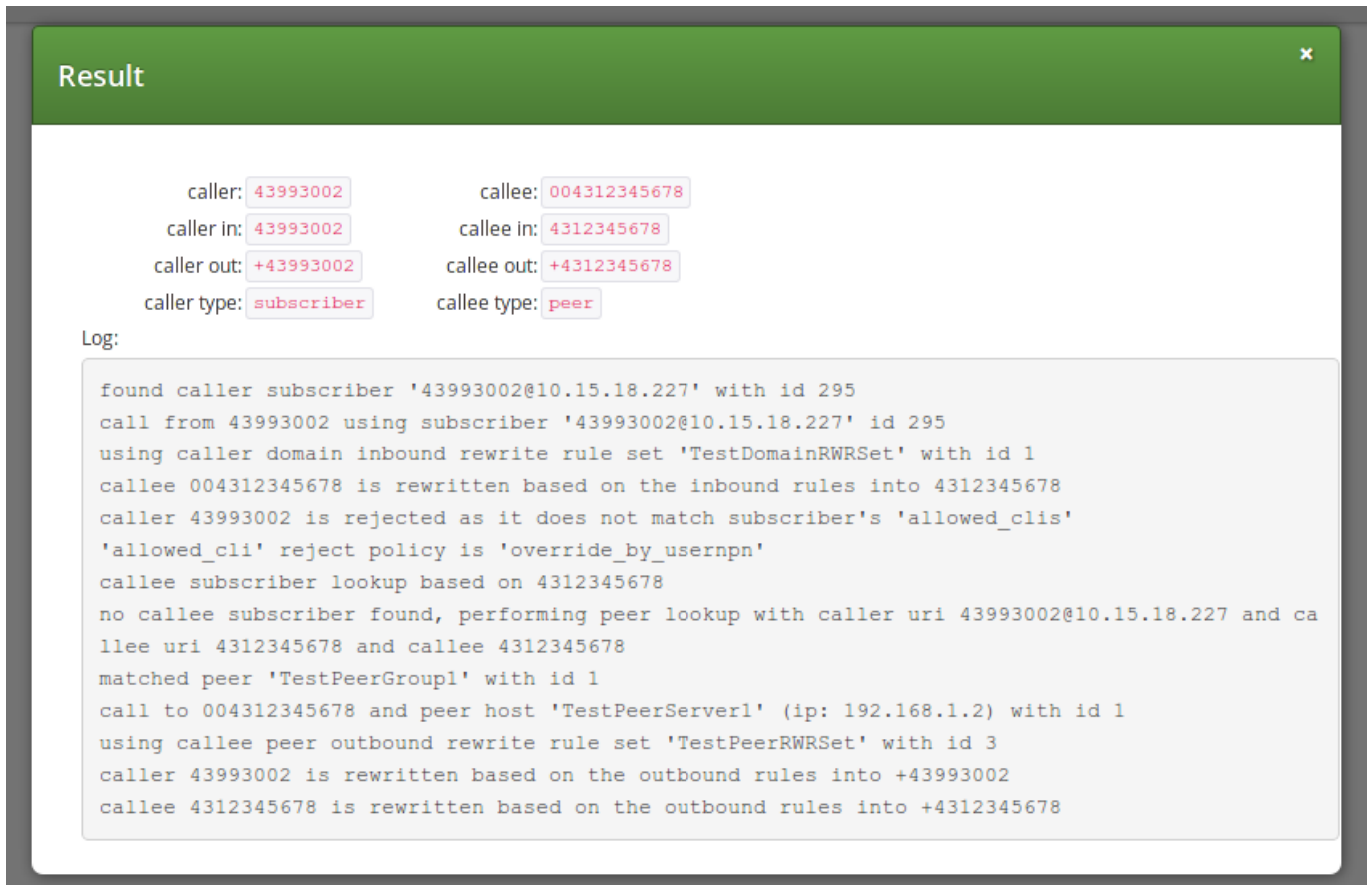
Showing 1 to 4 of 8 entries (filtered from 56 total entries) ← 1 2 →

Caller Rewrite Rules Override  
Callee Rewrite Rules Override  
Callee Peer Override

[Verify](#)

Figure 21: Call Routing Verif. - Only Numbers - Input

- Result:



The screenshot shows a 'Result' window with a green header and a close button. It displays call routing verification details for a call from 43993002 to 004312345678. The caller is identified as a 'subscriber' and the callee as a 'peer'. A log section provides a detailed step-by-step account of the routing process, including domain rewrite rules, subscriber lookups, and peer group matching.

caller: 43993002      callee: 004312345678  
caller in: 43993002      callee in: 4312345678  
caller out: +43993002      callee out: +4312345678  
caller type: subscriber      callee type: peer

Log:

```
found caller subscriber '43993002@10.15.18.227' with id 295
call from 43993002 using subscriber '43993002@10.15.18.227' id 295
using caller domain inbound rewrite rule set 'TestDomainRWRSet' with id 1
callee 004312345678 is rewritten based on the inbound rules into 4312345678
caller 43993002 is rejected as it does not match subscriber's 'allowed_clis'
'allowed_cli' reject policy is 'override_by_usernpn'
callee subscriber lookup based on 4312345678
no callee subscriber found, performing peer lookup with caller uri 43993002@10.15.18.227 and ca
llee uri 4312345678 and callee 4312345678
matched peer 'TestPeerGroup1' with id 1
call to 004312345678 and peer host 'TestPeerServer1' (ip: 192.168.1.2) with id 1
using callee peer outbound rewrite rule set 'TestPeerRWRSet' with id 3
caller 43993002 is rewritten based on the outbound rules into +43993002
callee 4312345678 is rewritten based on the outbound rules into +4312345678
```

Figure 22: Call Routing Verif. - Only Numbers - Result

## 2. Using phone number and URI, without explicit subscriber selection

- Input Data:

### Call Routing Verification

[← Back](#) [Expand Groups](#)

Caller number/uri

Callee number/uri

Caller Type  Subscriber  Peer

Caller Subscriber Search: .227

#	Username	Domain	UUID	Number	
295	43993002	10.15.18.227	51e32173-c8a9-44f1-af30-a1ed431eb2bf		<input type="checkbox"/>
297	43993003	10.15.18.227	6feb9ea-21c0-4f55-8828-80d546b8998f		<input type="checkbox"/>
299	43993004	10.15.18.227	3543a26e-861b-459f-a348-a8ef3e1e9eab		<input type="checkbox"/>
301	43993005	10.15.18.227	355773d2-1c08-475c-8858-eaf75bb58c73		<input type="checkbox"/>

Showing 1 to 4 of 8 entries (filtered from 56 total entries) ◀ 1 2 ▶

Caller Rewrite Rules Override

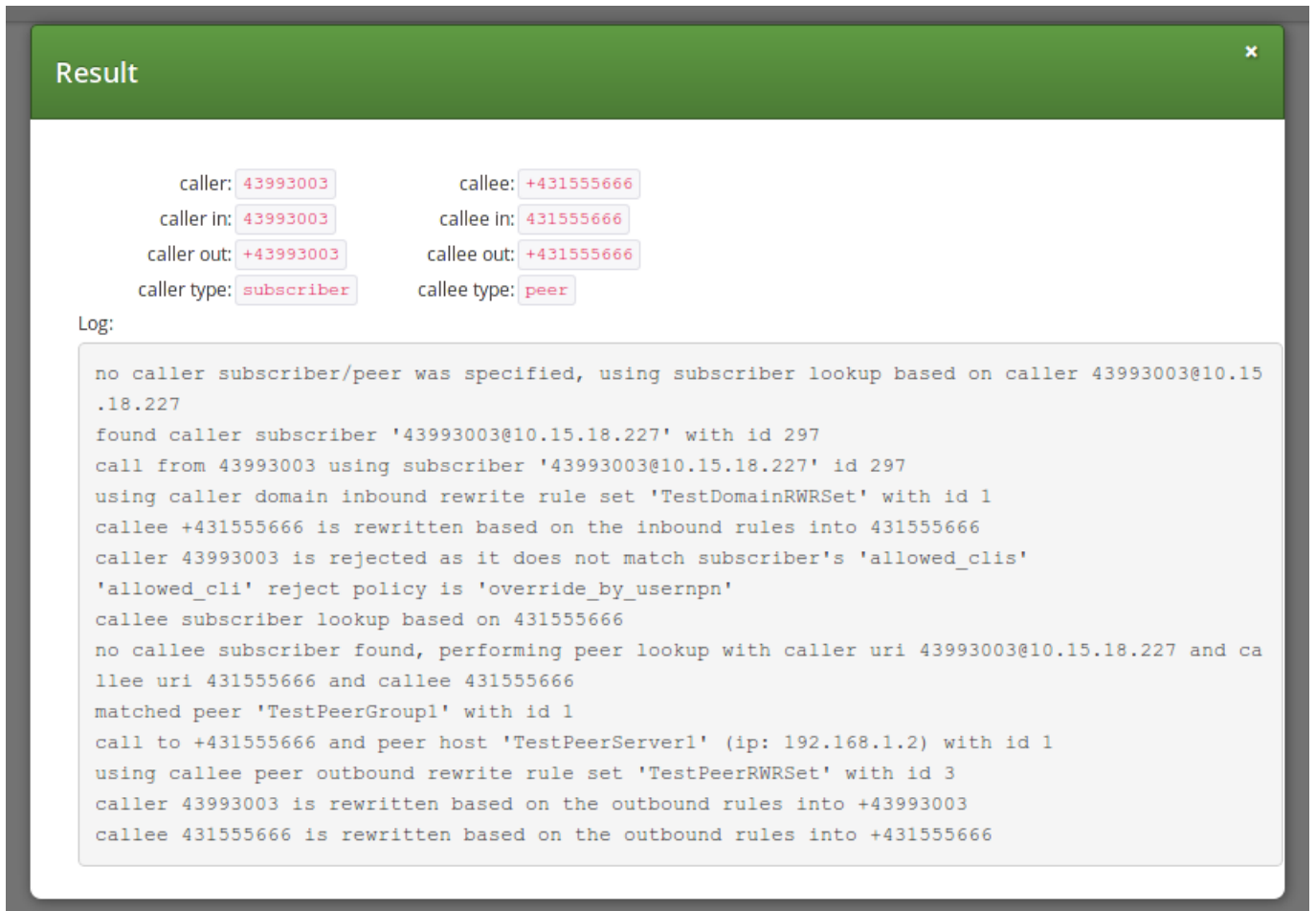
Callee Rewrite Rules Override

Callee Peer Override

[Verify](#)

Figure 23: Call Routing Verif. - Number and URI - Input

- Result:



The screenshot shows a window titled "Result" with a green header and a close button. Below the header, there are two columns of call details:

caller:	43993003	callee:	+431555666
caller in:	43993003	callee in:	431555666
caller out:	+43993003	callee out:	+431555666
caller type:	subscriber	callee type:	peer

Below the details is a "Log:" section containing the following text:

```
no caller subscriber/peer was specified, using subscriber lookup based on caller 43993003@10.15.18.227
found caller subscriber '43993003@10.15.18.227' with id 297
call from 43993003 using subscriber '43993003@10.15.18.227' id 297
using caller domain inbound rewrite rule set 'TestDomainRWRSet' with id 1
callee +431555666 is rewritten based on the inbound rules into 431555666
caller 43993003 is rejected as it does not match subscriber's 'allowed_clis'
'allowed_cli' reject policy is 'override_by_usernpn'
callee subscriber lookup based on 431555666
no callee subscriber found, performing peer lookup with caller uri 43993003@10.15.18.227 and callee uri 431555666 and callee 431555666
matched peer 'TestPeerGroup1' with id 1
call to +431555666 and peer host 'TestPeerServer1' (ip: 192.168.1.2) with id 1
using callee peer outbound rewrite rule set 'TestPeerRWRSet' with id 3
caller 43993003 is rewritten based on the outbound rules into +43993003
callee 431555666 is rewritten based on the outbound rules into +431555666
```

Figure 24: Call Routing Verif. - Number and URI - Result



## 5 Features

The sip:carrier provides plenty of subscriber features to offer compelling VoIP services to end customers, and also to cover as many deployment scenarios as possible. In this chapter, we provide the features overview and describe their function and use cases.

### 5.1 Managing System Administrators

The sip:carrier offers the platform operator with an easy to use interface to manage users with administrative privileges. Such users are representatives of resellers, and are entitled to manage configuration of services for *Customers*, *Subscribers*, *Domains*, *Billing Profiles* and other entities on Sipwise NGCP.

Administrators, as user accounts, are also used for client authentication on the REST API of NGCP.

There are two administrators, whose account is enabled by default. Both of them belong to the *default reseller*. These users are the *superusers* of the NGCP administrative web interface (the so-called "admin panel"), and they have the right to modify administrators of other *Resellers* as well. These users are:

- "administrator" is a default administrative account. It is fully manageable by the system owner.
- "sipwise" is solely for the Sipwise support access. This user can be only enabled or disabled but nor modified neither removed.

#### 5.1.1 Configuring Administrators

Configuration of access rights of system administrators is possible through the admin panel of NGCP. In order to do that, please navigate to *Settings* → *Administrators*.

**Administrators**

← Back   **★ Create Administrator**

Administrator successfully updated

Show 5 entries   Search:

#	Reseller	Login	Master	Active	Read Only	Show Passwords	Show CDRs	Show Billing Info	Lawful Intercept
1	default	administrator	1	1	0	1	1	1	1
3	Demo Reseller	demoadmin	1	1	0	1	1	0	0

Showing 1 to 2 of 2 entries

Navigation: ← 1 → ⇒

Figure 25: List of System Administrators

You have 2 options:

- If you'd like to **create** a new administrator user press *Create Administrator* button.
- If you'd like to **update** an existing administrator user press *Edit* button in its row.

There are some generic attributes that have to be set for each administrator:

The screenshot shows the 'Edit Administrator' form. At the top, there is a green header with the title 'Edit Administrator' and a close button. Below the header, there is a 'Reseller' section with a search bar. A table lists three resellers:

#	Name	Contract #	Status	
16	Demo Reseller	200	active	<input checked="" type="checkbox"/>
1	default	1	active	<input type="checkbox"/>
		137	active	<input type="checkbox"/>

Below the table, it says 'Showing 1 to 3 of 3 entries' and there are navigation buttons. To the right of the table is a 'Create Reseller' button. Below the table, there are input fields for 'Login' (containing 'demoadmin'), 'Password', and a checkbox for 'Is superuser'. At the bottom right, there is a 'Save' button.

Figure 26: Generic System Administrator Attributes

- *Reseller*: each administrator user must belong to a *Reseller*. There is always a default reseller (ID: 1, Name: default), but the administrator has to be assigned to his real reseller, if such an entity (other than default) exists.
- *Login*: the login name of the administrator user
- *Password*: the password of the administrator user for logging in the admin panel, or for authentication on REST API

The second set of attributes is a list of access rights that are discussed in subsequent section of the handbook.

### 5.1.2 Access Rights of Administrators

The various access rights of administrators are shown in the figure and summarized in the table below.

The screenshot shows a web form titled "Edit Administrator" with a close button (X) in the top right corner. The form contains several checkboxes for access rights:

- Is superuser:
- Is master:
- Is active:
- Read only:
- Show passwords:
- Call data:
- Billing data:
- Lawful Intercept:

A "Save" button is located at the bottom right of the form.

Figure 27: Access Rights of System Administrators

Table 1: Access Rights of System Administrators

Label in admin list	Access Right	Description
<i>not shown</i>	Is superuser	The user is allowed to modify data on Reseller level and — among others — is able to modify administrators of other resellers. There should be only 1 user on Sipwise NGCP with this privilege.
Master	Is master	The user is allowed to create, delete or modify other Admins who belong to the same Reseller.
Active	Is active	The user account is active, i.e. the admin user can login on the web panel or authenticate himself on REST API; otherwise user authentication will fail.

Table 1: (continued)

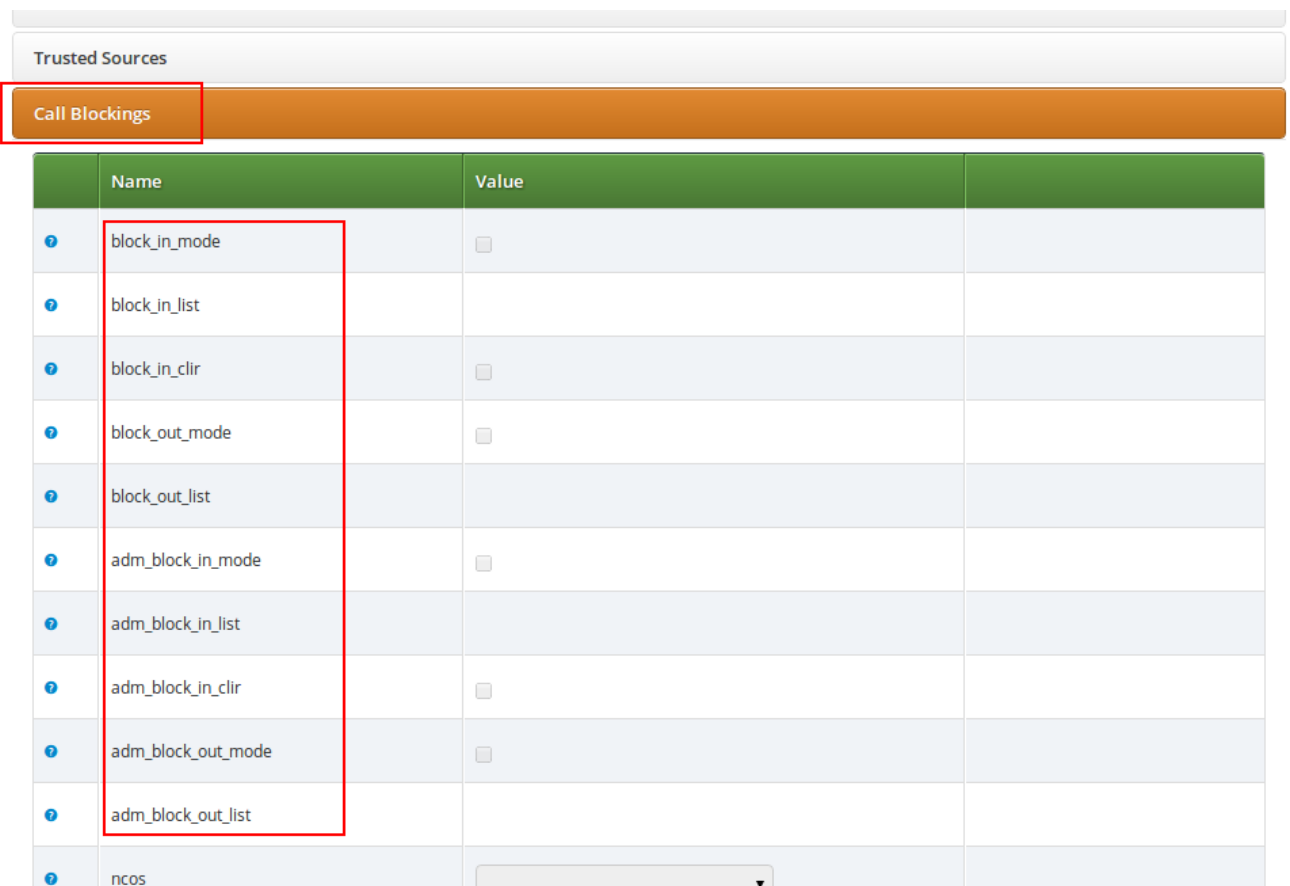
Label in admin list	Access Right	Description
Read Only	Read only	<p>The user will only be able to list various data but is not allowed to modify anything.</p> <ul style="list-style-type: none"> <li>For the <b>web interface</b> this means that <i>Create...</i> and <i>Edit</i> buttons will be hidden or disabled.</li> <li>For the <b>REST API</b> this means that only <code>GET</code>, <code>HEAD</code>, <code>OPTIONS</code> HTTP request methods are accepted, and NGCP will reject those targeting data modification: <code>PUT</code>, <code>PATCH</code>, <code>POST</code>, <code>DELETE</code>.</li> </ul>
Show Passwords	Show passwords	<p>The user sees subscriber passwords (in plain text) on the web interface.</p> <hr/> <p><b>Note</b></p> <p>Admin panel user passwords are stored in an unreadable way (cryptographic hash digest) in the database, while subscriber passwords are basically always stored in plain text. The latter happens on purpose, e.g. to make subscriber data migration possible.</p> <hr/>
Show CDRs	Call data	<p>This privilege has effect on 2 items that will be displayed on admin panel of NGCP, when <i>Subscriber</i> → <i>Details</i> is selected:</p> <ol style="list-style-type: none"> <li>1. <i>PBX Groups</i> list</li> <li>2. <i>Captured Dialogs</i> list</li> </ol>
Show Billing Info	Billing data	<p>Some REST API resources that are related to billing are disabled: HTTP requests on <code>/api/vouchers</code>, <code>/api/topupcash</code> and <code>/api/topupvoucher</code> resources are rejected.</p>
Lawful Intercept	Lawful intercept	<p>If the privilege is selected then the REST API for interceptions (that is: <code>/api/interceptions</code>) is enabled; if the privilege is not selected then the interceptions API is disabled.</p> <hr/> <p><b>Note</b></p> <p>This means that besides enabling LI in <code>config.yml</code> configuration file one also needs to enable the API via the LI privilege of an administrator user, so that NGCP can really provide LI service.</p> <hr/>

## 5.2 Access Control for SIP Calls

There are two different methods to provide fine-grained call admission control to both subscribers and admins. One is *Block Lists*, where you can define which numbers or patterns can be called from a subscriber to the outbound direction and which numbers or patterns are allowed to call a subscriber in the inbound direction. The other is *NCOS Levels*, where the admin predefines rules for outbound calls, which are grouped in certain levels. The subscriber can then just choose the level, or the admin can restrict a subscriber to a certain level. Also sip:carrier offers some options to restrict the IP addresses that subscriber is allowed to use the service from. The following sections describe these features in detail.

### 5.2.1 Block Lists

*Block Lists* provide a way to control which users/numbers can call or be called, based on a subscriber level, and can be found in the *Call Blockings* section of the subscriber preferences.



	Name	Value
?	block_in_mode	<input type="checkbox"/>
?	block_in_list	
?	block_in_clir	<input type="checkbox"/>
?	block_out_mode	<input type="checkbox"/>
?	block_out_list	
?	adm_block_in_mode	<input type="checkbox"/>
?	adm_block_in_list	
?	adm_block_in_clir	<input type="checkbox"/>
?	adm_block_out_mode	<input type="checkbox"/>
?	adm_block_out_list	
?	ncos	<input type="text"/>

Block Lists are separated into *Administrative Block Lists* (*adm\_block\_\**) and *Subscriber Block Lists* (*block\_\**). They both have the same behaviour, but Administrative Block Lists take higher precedence. Administrative Block Lists are only accessible by the system administrator and can thus be used to override any Subscriber Block Lists, e.g. to block certain destinations. The following break-down of the various block features apply to both types of lists.

### 5.2.1.1 Block Modes

Block lists can either be *whitelists* or *blacklists* and are controlled by the User Preferences *block\_in\_mode*, *block\_out\_mode* and their administrative counterparts.

- The *blacklist* mode (option is not checked) tells the system to **allow anything except the entries in the list**. Use this mode if you just want to block certain numbers and allow all the rest.
- The *whitelist* mode indicates to **reject anything except the entries in the list**. Use this mode if you want to enforce a strict policy and allow only selected destinations or sources.

You can change a list mode from one to the other at any time.

### 5.2.1.2 Block Lists

The list contents are controlled by the User Preferences *block\_in\_list*, *block\_out\_list* and their administrative counterparts. Click on the *Edit* button in the *Preferences* view to define the list entries.

In block list entries, you can provide shell patterns like `*` and `[]`. The behavior of the list is controlled by the *block\_xxx\_mode* feature (so they are either allowed or rejected). In our example above we have *block\_out\_mode* set to *blacklist*, so all calls to US numbers and to the Austrian number +431234567 are going to be rejected.

The screenshot shows the 'Edit Preference block\_out\_list' dialog box in the sip:wise interface. The dialog has a green header with the title 'Edit Preference block\_out\_list' and a close button (X) in the top right corner. Below the header, there are two input fields. The first field contains '1\*' and has a red '1' next to it. The second field contains '431234567' and has a red '2' next to it. To the right of these fields are two buttons: a trash can icon and a circular arrow icon. Below these buttons is an 'Add' button. The background of the interface is dimmed, showing a sidebar with 'Call Blockings' selected and a table with columns 'Name' and 'Value'.

Click the *Close* icon once you're done editing your list.

### 5.2.1.3 Block Anonymous Numbers

For incoming call, the User Preference *block\_in\_clir* and *adm\_block\_in\_clir* controls whether or not to reject incoming calls with number suppression (either "[Aa]nonymous" in the display- or user-part of the From-URI or a header *Privacy: id* is set). This flag is independent from the Block Mode.

### 5.2.2 NCOS Levels

*NCOS Levels* provide predefined lists of allowed or denied destinations for outbound calls of local subscribers. Compared to *Block Lists*, they are much easier to manage, because they are defined on a global scope, and the individual levels can then be assigned to each subscriber. Again there is the distinction for user- and administrative-levels.

If case of a conflict, when the Block Lists feature allows a number and NCOS Levels rejects the same number or vice versa, the number will be rejected.

NCOS levels can either be *whitelists* or *blacklists*.

- The *blacklist* mode indicates to **allow everything except the entries in this level**. This mode is used if you want to just block certain destinations and allow all the rest.
- The *whitelist* mode indicates to **reject anything except the entries in this level**. This is used if you want to enforce a strict policy and allow only selected destinations.

#### 5.2.2.1 Creating NCOS Levels

To create an NCOS Level, go to *Settings*→*NCOS Levels* and press the *Create NCOS Level* button.

The screenshot shows the sip:wise NGCP Dashboard interface. At the top right, it indicates the user is logged in as 'administrator' with a 'Logout' link. The dashboard title is 'sip:wise NGCP Dashboard' and includes a 'Settings' menu. The main section is titled 'NCOS Levels'. Below this, there are two buttons: 'Back' and 'Create NCOS Level', with the latter being highlighted by a red rectangular box. To the right of these buttons is a search input field. Below the buttons is a table with the following columns: '#', 'Reseller', 'Level Name', 'Mode', and 'Description'. The table is currently empty, displaying the message 'No data available in table'. Below the table, it shows 'Showing 0 to 0 of 0 entries' and a set of navigation arrows. At the bottom of the dashboard, there is a copyright notice: '© 2013 Sipwise GmbH, all rights reserved.'

Select a reseller, enter a name, select the mode and add a description, then click the Save button.



The screenshot shows the 'Create NCOS Levels' dialog in the sip:wise interface. The dialog is titled 'Create NCOS Levels' and has a close button (X) in the top right corner. It features a 'Reseller' section with a search bar and a table of resellers. The table has columns for '#', 'Name', 'Contract #', 'Status', and a checkbox. The first row shows a reseller with ID 1, Name 'default', Contract # 1, and Status 'active'. The checkbox for this reseller is checked and highlighted with a red box. Below the table, it says 'Showing 1 to 1 of 1 entries' and has navigation buttons. A 'Create Reseller' button is also present. The main form has three fields: 'Level Name' with the value 'test' (highlighted with a red box and a red '2'), 'Mode' with the value 'blacklist' (highlighted with a red box and a red '3'), and 'Description' with the value 'NCOS Test Level' (highlighted with a red box and a red '4'). A 'Save' button is at the bottom right, highlighted with a red box and a red '5'. The background shows the sip:wise interface with a 'Logged in as administrator' status and a 'Logout' link.

#	Name	Contract #	Status	
1	default	1	active	1 <input checked="" type="checkbox"/>

Showing 1 to 1 of 1 entries

Level Name: test 2

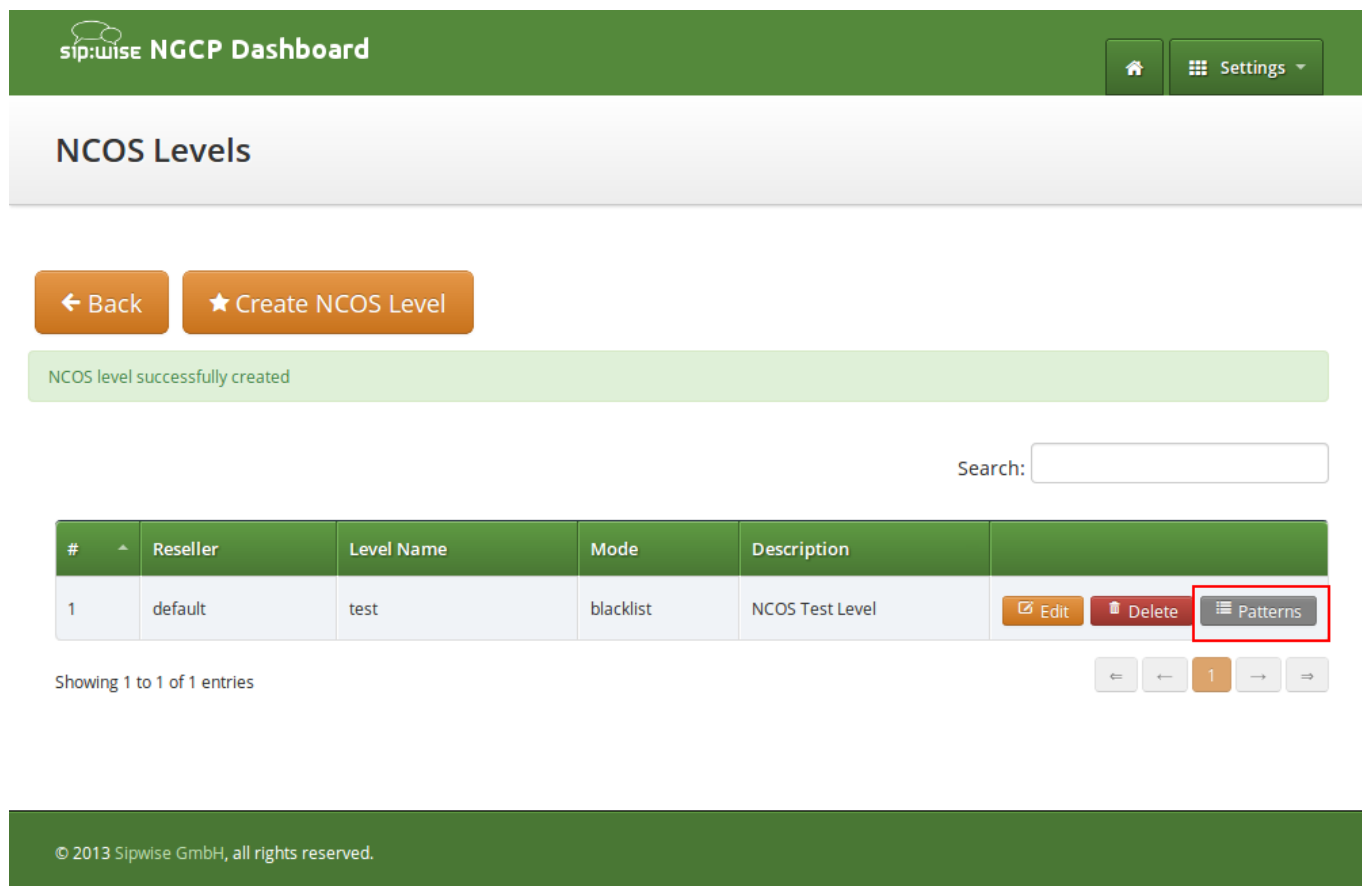
Mode: blacklist 3

Description: NCOS Test Level 4

Save 5

### 5.2.2.2 Creating Rules per NCOS Level

To define the rules within the newly created NCOS Level, click on the *Patterns* button of the level.



sip:wise NGCP Dashboard

NCOS Levels

← Back   ★ Create NCOS Level

NCOS level successfully created

Search:

#	Reseller	Level Name	Mode	Description	
1	default	test	blacklist	NCOS Test Level	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Patterns</a>

Showing 1 to 1 of 1 entries

© 2013 Sipwise GmbH, all rights reserved.

There are 2 groups of patterns where you can define matching rules for the selected NCOS Level:

- NCOS Number Patterns: here you can define number patterns that will be matched against the called number and allowed or blocked, depending on whitelist / blacklist mode. The patterns are regular expressions.
- NCOS LNP Carriers: here you can select predefined *LNP Carriers* that will be allowed (whitelist mode) or prohibited (blacklist mode) to route calls to them. (See Section 5.4.1 in the handbook for the description of LNP functionality)

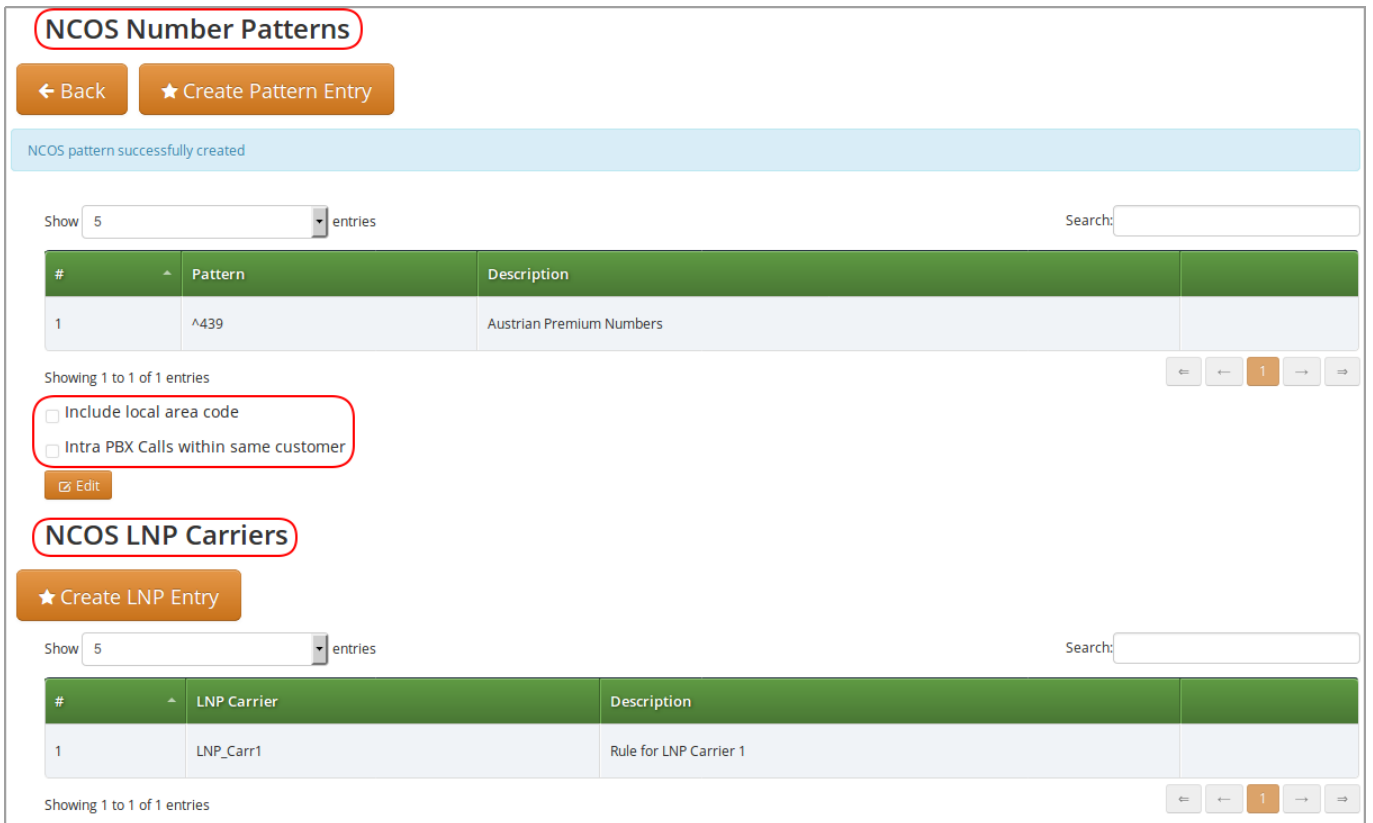


Figure 28: NCOS Patterns List

In the **NCOS Number Patterns** view you can create multiple patterns to define your level, one after the other. Click on the *Create Pattern Entry* Button on top and fill out the form.

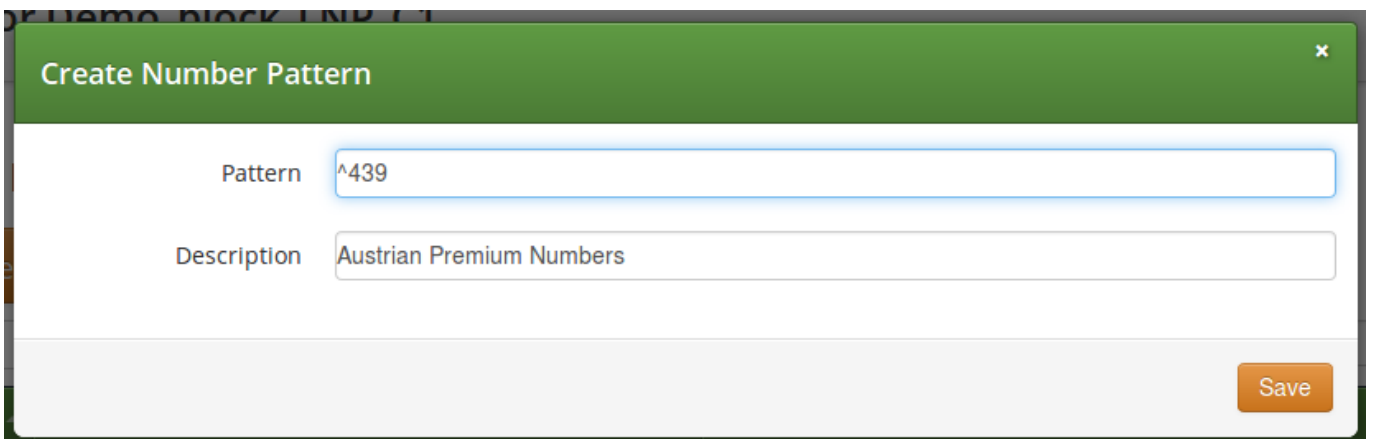


Figure 29: Create NCOS Number Pattern

In this example, we block (since the mode of the level is *blacklist*) all numbers starting with 439. Click the *Save* button to save the entry in the level.

There are **2 options** that help you to easily define specific number ranges that will be allowed or blocked, depending on whitelist / blacklist mode:

- *Include local area code*: all subscribers within the caller's local area, e.g. if a subscriber has country-code 43 and area-code 1, then selecting this checkbox would result in the implicit number pattern:  $\wedge 431$ .
- *Intra PBX calls within same customer*: all subscribers that belong to the same PBX customer as the caller himself.

In the **NCOS LNP Carriers** view you can select specific LNP Carriers—i.e. carriers that host the called ported numbers—that will be allowed or blocked for routing calls to them (whitelist / blacklist mode, respectively).

Sipwise NGCP performs number matching always with the dialed number and not with the number generated after LNP lookup that is: either the original dialed number prefixed with an LNP carrier code, or the routing number.

An example of *NCOS LNP Carrier* pattern definition:

The screenshot shows a web interface titled "Create LNP Carriers". It features a search bar, a table of LNP carriers, and a description field. The table has columns for ID, Name, Prefix, and a selection checkbox. The entry "LNP\_Carr1" is selected. Below the table are pagination controls and a "Create LNP Carrier" button. The description field contains "Rule for LNP Carrier 1" and a "Save" button is at the bottom right.

#	Name	Prefix	
11	test_lnp_carrier_4_1510288861	test1510288861	<input type="checkbox"/>
13	test_lnp_carrier_5_1510288862	test1510288862	<input type="checkbox"/>
15	test_lnp_carrier_6_1510288863	test1510288863	<input type="checkbox"/>
17	LNP_Carr1	C1	<input checked="" type="checkbox"/>

Showing 5 to 8 of 9 entries

Create LNP Carrier

Description: Rule for LNP Carrier 1

Save

Figure 30: Create NCOS LNP Carrier

In the above example we created a rule that blocks calls to "LNP\_Carr1" carrier, supposing we use blacklist mode of the NCOS Level.

#### Note

Currently NGCP does not support filtering of individual phone numbers in addition to LNP Carrier matching. In other words: combining phone number and LNP Carrier patterns is not possible.

**Tip**

There might be situations when phone number patterns may not be strictly aligned with telephony providers, for instance in case of full number portability in a country. In such cases using *NCOS LNP Carriers* patterns still allows for defining NCOS levels that allow / block calls to mobile numbers, for example. In order to achieve this goal you have to list all LNP carriers in the NCOS patterns that are known to host mobile numbers.

**5.2.2.3 Assigning NCOS Levels to Subscribers/Domains**

Once you’ve defined your NCOS Levels, you can assign them to local subscribers. To do so, navigate to *Settings*→*Subscribers*, search for the subscriber you want to edit, press the *Details* button and go to the *Preferences* View. There, press the *Edit* button on either the *ncos* or *adm\_ncos* setting in the *Call Blockings* section.

Call Blockings	
Name	Value
block_in_mode	<input type="checkbox"/>
block_in_list	
block_in_clir	<input type="checkbox"/>
block_out_mode	<input type="checkbox"/>
block_out_list	1* 431234567
adm_block_in_mode	<input type="checkbox"/>
adm_block_in_list	
adm_block_in_clir	<input type="checkbox"/>
adm_block_out_mode	<input type="checkbox"/>
adm_block_out_list	
ncos	<input type="text" value=""/>

**Edit**

You can assign the NCOS level to all subscribers within a particular domain. To do so, navigate to *Settings*→*Domains*, select the domain you want to edit and click *Preferences*. There, press the *Edit* button on either *ncos* or *admin\_ncos* in the *Call Blockings* section.

Note: if both domain and subscriber have same NCOS preference set (either *ncos* or *adm\_ncos*, or both) the subscriber’s preference is used. This is done so that you can override the domain-global setting on the subscriber level.

### 5.2.2.4 Assigning NCOS Level for Forwarded Calls to Subscribers/Domains

In some countries there are regulatory requirements that prohibit subscribers from forwarding their numbers to special numbers like emergency, police etc. While the sip:carrier does not deny provisioning Call Forward to these numbers, the administrator can prevent the incoming calls from being actually forwarded to numbers defined in the NCOS list: just select the appropriate NCOS level in the domain's or subscriber's preference *adm\_cf\_ncos*. This NCOS will apply only to the Call Forward from the subscribers and not to the normal outgoing calls from them.

### 5.2.3 IP Address Restriction

The sip:carrier provides subscriber preference *allowed\_ips* to restrict the IP addresses that subscriber is allowed to use the service from. If the REGISTER or INVITE request comes from an IP address that is not in the allowed list, the sip:carrier will reject it with a 403 message. Also a voice message can be played when the call attempt is rejected (if configured).

By default, *allowed\_ips* is an empty list which means that subscriber is not restricted. If you want to configure a restriction, navigate to *Settings*→*Subscribers*, search for the subscriber you want to edit, press *Details* and then *Preferences* and press *Edit* for the *allowed\_ips* preference in the *Access Restrictions* section.

Call Blockings			
Access Restrictions			
	Name	Value	
1	lock		
	concurrent_max		
	concurrent_max_out		
	allowed_clis		
	reject_emergency	<input type="checkbox"/>	
	concurrent_max_per_account		
	concurrent_max_out_per_account		
	allowed_ips		3 <input type="button" value="Edit"/>
	man_allowed_ips		
	ignore_allowed_ips	<input type="checkbox"/>	
	allow_out_foreign_domain	<input type="checkbox"/>	

Press the Edit button to the right of empty drop-down list.

You can enter multiple allowed IP addresses or IP address ranges one after another. Click the *Add* button to save each entry in the list. Click the *Delete* button if you want to remove some entry.

## 5.3 Call Forwarding and Call Hunting

The sip:carrier provides the capabilities for normal *call forwarding* (deflecting a call for a local subscriber to another party immediately or based on events like the called party being busy or doesn't answer the phone for a certain number of seconds) and *serial call hunting* (sequentially executing a group of deflection targets until one of them succeeds). Targets can be stacked, which means if a target is also a local subscriber, it can have another call forward or hunt group which is executed accordingly.

Call Forwards and Call Hunting Groups can either be executed unconditionally or based on a *Time Set Definition*, so you can define deflections based on time period definitions (e.g. Monday to Friday 8am to 4pm etc).

### 5.3.1 Setting a simple Call Forward

Go to your *Subscriber Preferences* and click *Edit* on the Call Forward Type you want to set (e.g. *Call Forward Unconditional*).

The screenshot shows the 'Edit Call Forward Unconditional' dialog box. It has a green header with a close button. The main content area is white and contains the following elements:

- Destination:** Three radio buttons: 'Voicemail', 'Conference', and 'URI/Number'. The 'URI/Number' option is selected and highlighted with a red box labeled '1'.
- URI/Number:** A text input field containing '4312345', highlighted with a red box labeled '2'.
- for (seconds):** A text input field containing '300'.
- Buttons:** 'Advanced View' and 'Save' buttons at the bottom right. The 'Save' button is highlighted with a red box labeled '3'.

The background shows a table with columns for 'Type', 'Call Forward', and 'Call Forward Timeout'. Below the table is a section for 'Voicemail and Voicebox'.

If you select *URI/Number* in the *Destination* field, you also have to set a *URI/Number*. The timeout defines for how long this destination should be tried to ring.

### 5.3.2 Advanced Call Hunting

Beside call forwarding to a single destination, Sipwise NGCP offers the possibility to activate call forwarding in a more sophisticated way:

- to multiple destinations (→ *Destination Set*)
- only during a pre-defined time set (→ *Time Set*)
- only for specific callers (→ *Source Set*)

If you want to define such more detailed call forwarding rules, you need to change into the *Advanced View* when editing your call forward. There, you can select multiple *Destination Set - Time Set - Source Set* triples that determine all conditions under which the call will be forwarded.

### Explanation of call forward parameters

- A **Destination Set** is a list of destinations where the call will be routed to, one after another, according to the order of their assigned priorities. See the [Destination Sets](#) Section 5.3.2.1 subchapter for a detailed description.
- A **Time Set** is a time period definition, i.e. when the call forwarding has to be active. See the [Time Sets](#) Section 5.3.2.2 subchapter for a detailed description.
- A **Source Set** is a list of number patterns that will be matched against the calling party number; if the calling number matches the call forwarding will be executed. See the [Source Sets](#) Section 5.3.2.3 subchapter for a detailed description.

#### 5.3.2.1 Configuring Destination Sets

Click on *Manage Destination Sets* to see a list of available sets. The *quickset\_cfu* has been implicitly created during our creation of a simple call forward. You can edit it to add more destinations, or you can create a new destination set.

1 Name my test set

Destination  Voicemail  URI/Number

2 URI/Number 12345

for (seconds) 300

4 Priority 1

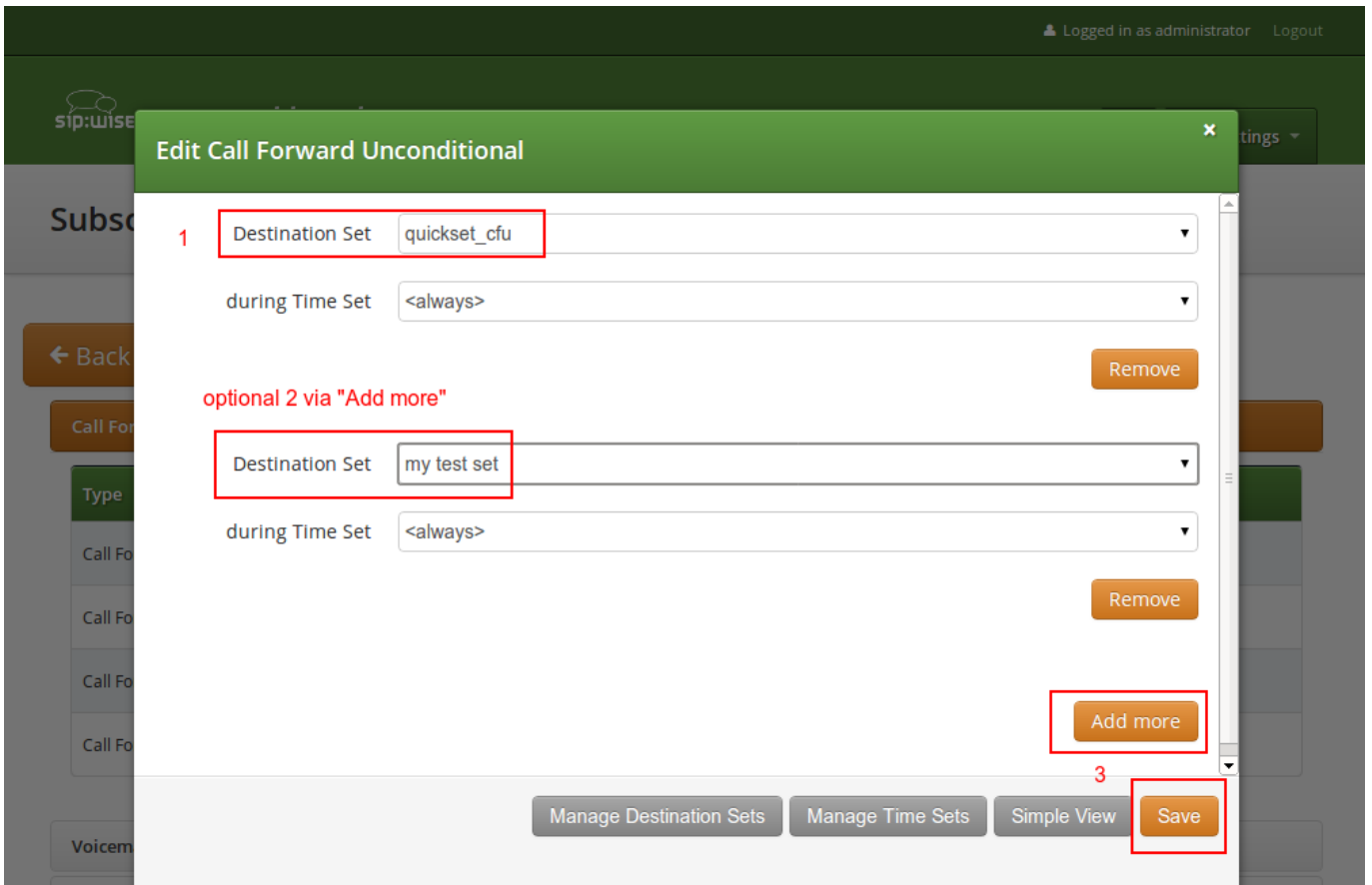
optional 5 Add another destination

6 Save

Remove



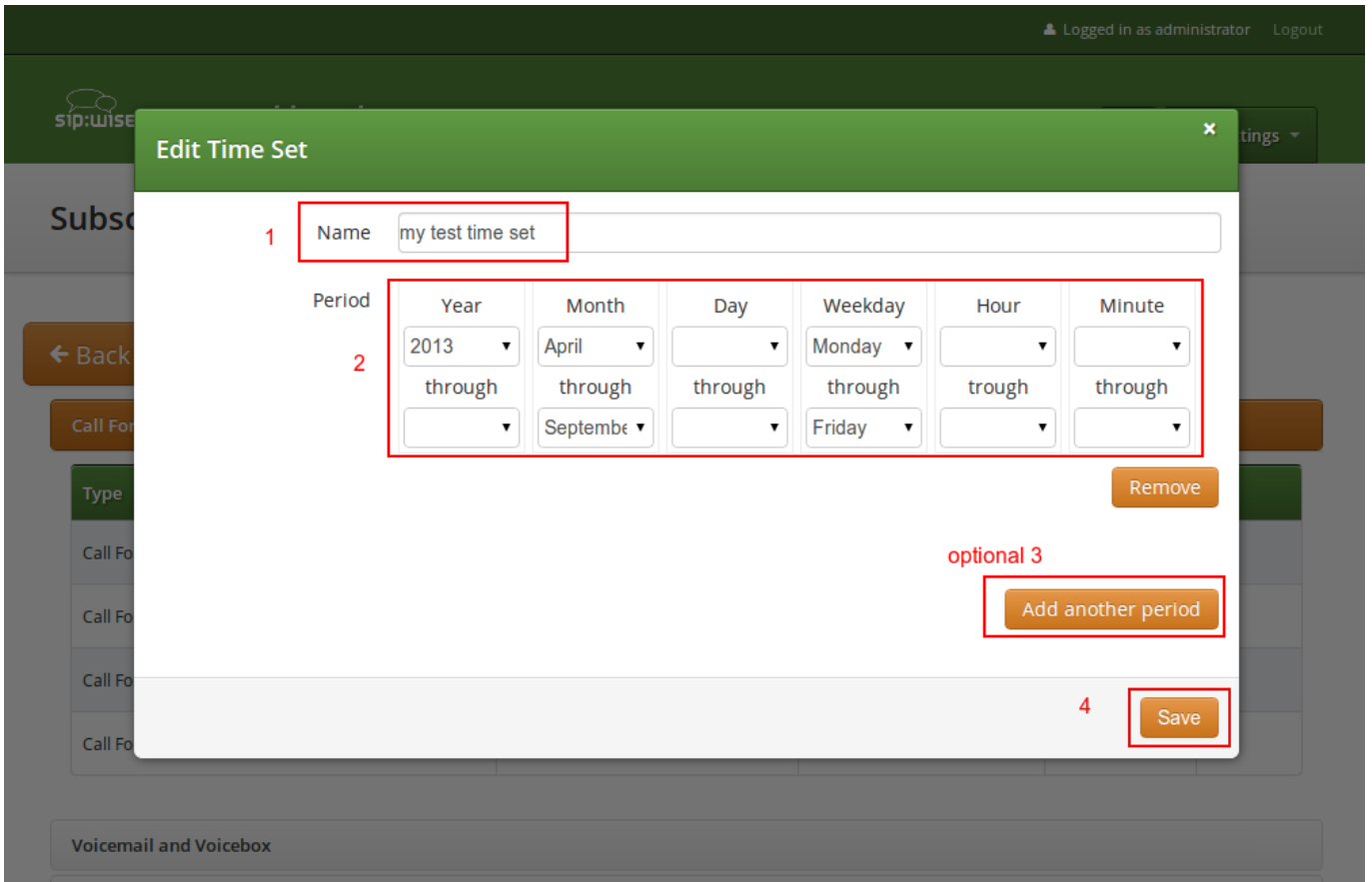
When you close the *Destination Set Overview*, you can now assign your new set in addition or instead of the *quickset\_cfu* set.



Press *Save* to store your settings.

### 5.3.2.2 Configuring Time Sets

Click on *Manage Time Sets* in the advanced call-forward menu to see a list of available time sets. By default there are none, so you have to create one.



You need to provide a *Name*, and a list of *Periods* where this set is active. If you only set the top setting of a date field (like the *Year* setting in our example above), then it's valid for just this setting (like the full year of *2013* in our case). If you provide the bottom setting as well, it defines a period (like our *Month* setting, which means from beginning of April to end of September). For example, if a CF is set with the following timeset: "hour { 10-12 } minute { 20-30 }", the CF will be matched within the following time ranges:

- from 10.20am to 10:30am
- from 11.20am to 11:30am
- from 12.20am to 12:30am



**Important**

the period is a *through* definition, so it covers the full range. If you define an *Hour* definition *8-16*, then this means from *08:00* to *16:59:59* (unless you filter the *Minutes* down to something else).

If you close the *Time Sets* management, you can assign your new time set to the call forwards you're configuring.

**5.3.2.3 Configuring Source Sets**

Once the *Advanced View* of the call forward definition has been opened, you will need to press the *Manage Source Sets* button to start defining new Source Sets or managing an existing one. The following image shows the Source Set definition dialog:

**Edit Source Set** ✕

Name

Source  Remove

Source  Remove

Add another source

Save

Figure 31: Creating a Call Forward Source Set

You will need to fill in the `Name` field first, then in `Source` field you can enter:

- A simple phone number in E.164 format
- A pattern, in order to define a range of numbers. You can use "\*" (matches a string of 0 to any number of characters), "?" (matches any single character), "[abc]" (matches a single character that is part of the explicitly listed set: a, b or c) and "[0-9]" (matches a single character that falls in the range 0 to 9) as wildcards, as usual in shell patterns. Examples:
  - "431\*" (all numbers from Vienna / Austria)
  - "49176[0-5]77\*" (German numbers containing fixed digits and a variable digit in 0-5 range in position 6)
  - "43130120??" (numbers from Vienna with fixed prefix and 2 digits variable at the end)
- The constant string "anonymous" that indicates a suppressed calling number (CLIR)

You can add more patterns to the Source Set by pressing the *Add another source* button. When you finished adding all patterns, press the *Save* button. You will then see the below depicted list of Source Sets:

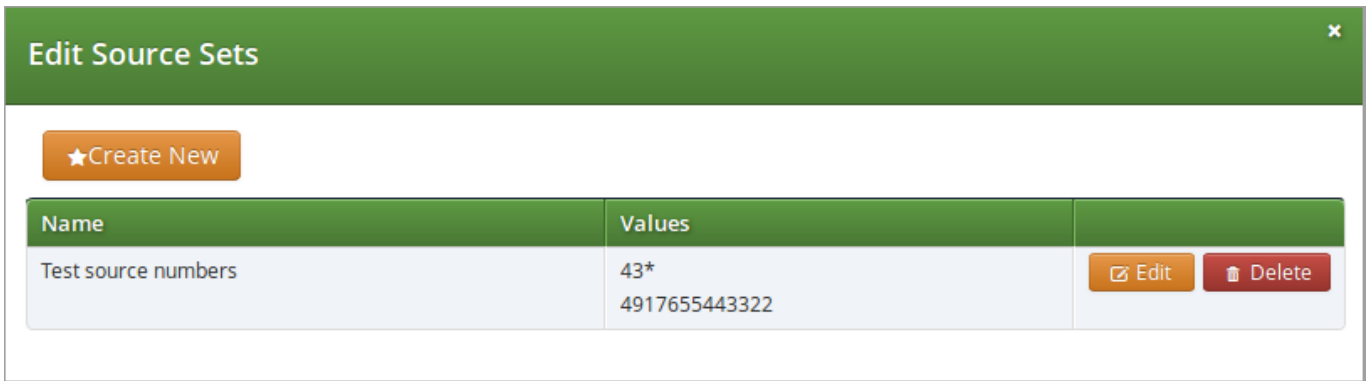


Figure 32: List of Call Forward Source Sets

As a next step you can define a Destination Set as described in [Destination Sets](#) Section 5.3.2.1 subchapter. For our example, we have defined the following Destination Set:

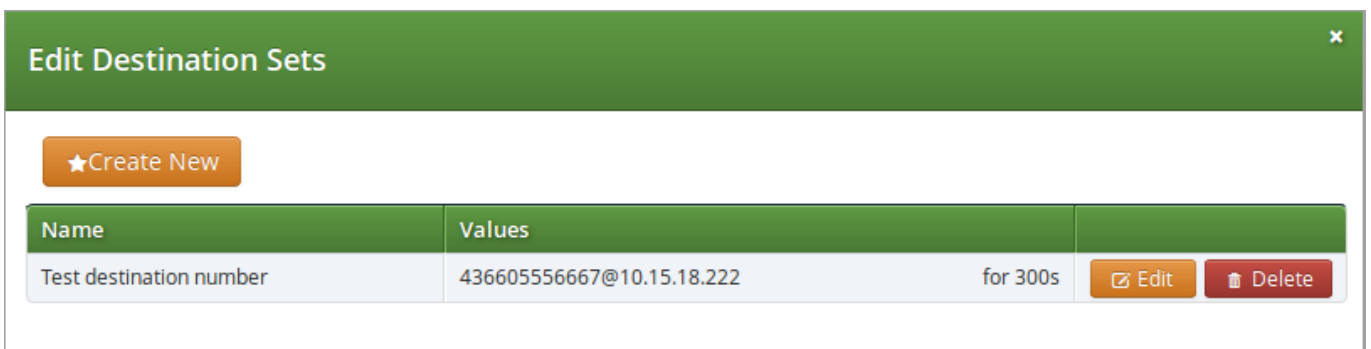


Figure 33: List of Call Forward Destination Sets

A final step of defining the call forward settings is selecting a Destination, a Time and a Source Set, as shown in the image below. *Please note* that there is no specific Time Set selected in our example, that means the call forward rule is valid (as shown) <always>.

Figure 34: Definition of a Call Forward with Source and Destination Sets

Once all the settings have been defined and the changes are saved, you will see the call forward entry (in our example: *Call Forward Unconditional*), with the names of the selected Destination, Time and Source Sets provided, at *Subscriber Preferences* → *Call Forwards* location on the web interface:

Subscriber Preferences for 43993005@10.15.18.222

← Back Expand Groups

Successfully saved Call Forward

Call Forwards

Type	Answer Timeout	Destinations	Timeset	Sources	
Call Forward Unconditional		Test destination number	always	Test source numbers	<span>Edit</span> <span>Delete</span>
Call Forward Busy					
Call Forward Timeout					

Figure 35: List of Call Forward with Source and Destination Sets

## 5.4 Local Number Porting

The Sipwise NGCP platform comes with two ways of accomplishing local number porting (LNP):

- one is populating the integrated LNP database with porting data,
- the other is accessing external LNP databases via the Sipwise LNP daemon using the LNP API.

---

### Note

Accessing external LNP databases is available for PRO and CARRIER products only.

---

### 5.4.1 Local LNP Database

The local LNP database provides the possibility to define LNP Carriers (the owners of certain ported numbers or number blocks) and their corresponding LNP Numbers belonging to those carriers. It can be configured on the admin panel in *Settings*→*Number Porting* or via the API. The LNP configuration can be populated individually or via CSV import/export both on the panel and the API.

#### 5.4.1.1 LNP Carriers

LNP Carriers are defined by an arbitrary *Name* for proper identification (e.g. *British Telecom*) and contain a *Prefix* which can be used as routing prefix in LNP Rewrite Rules and subsequently in Peering Rules to route calls to the proper carriers. The LNP prefix is written to CDRs to identify the selected carrier for post processing and analytics purposes of CDRs. LNP Carrier entries also have an *Authoritative* flag indicating that the numbers in this block belong to the carrier operating the sip:carrier. This is useful to define your own number blocks, and in case of calls to those numbers reject the calls if the numbers are not assigned to local subscribers (otherwise they would be routed to a peer, which might cause call loops). Finally the *Skip Rewrite* flag skips executing of LNP Rewrite Rules if no number manipulation is desired for an LNP carrier.

#### 5.4.1.2 LNP Numbers

LNP Carriers contain one or more LNP Numbers. Those LNP Numbers are defined by a *Number* entry in E164 format (*<cc><ac><sn>*) used to match a number against the LNP database. Number matching is performed on a longest match, so you can define number blocks without specifying the full subscriber number (e.g. a called party number *431999123* is going to match an entry *431999* in the LNP Numbers).

For an LNP Numbers entry, an optional *Routing Number* can be defined. This is useful to translate e.g. premium 900 or toll-free 800 numbers to actual routing numbers. If a Routing Number is defined, the called party number is implicitly replaced by the Routing Number and the call processing is continued with the latter. For external billing purposes, the optional *Type* tag of a matched LNP number is recorded in CDRs.

An optional *Start Date* and *End Date* allows to schedule porting work-flows up-front by populating the LNP database with certain dates, and the entries are only going to become active with those dates. Empty values for start indicate a start date in the past, while empty values for end indicate an end time in the future during processing of a call, allowing to define infinite date ranges. As intervals can overlap, the LNP number record with a start time closest to the current time is selected.

### 5.4.1.3 Enabling local LNP support

In order to activate Local LNP during routing, the feature must be activated in *config.yml*. Set *kamailio→proxy→lnp→enabled* to *yes* and *kamailio→proxy→lnp→type* to *local*.

### 5.4.1.4 LNP Routing Procedure

#### ***Calls to non-authoritative Carriers***

When a call arrives at the system, the calling and called party numbers are first normalized using the *Inbound Rewrite Rules for Caller* and *Inbound Rewrite Rules for Callee* within the rewrite rule set assigned to the calling party (a local subscriber or a peer).

If the called party number is not assigned to a local subscriber, or if the called party is a local subscriber and has the subscriber/-domain preference *lnp\_for\_local\_sub* set, the LNP lookup logic is engaged, otherwise the call proceeds without LNP lookup. The further steps assume that LNP is engaged.

If the call originated from a peer, and the peer preference *caller\_lnp\_lookup* is set for this peer, then an LNP lookup is performed using the normalized calling party number. The purpose for that is to find the LNP prefix of the calling peer, which is then stored as *source\_lnp\_prefix* in the CDR, together with the selected LNP number's *type* tag (*source\_lnp\_type*). If the LNP lookup does not return a result (e.g. the calling party number is not populated in the local LNP database), but the peer preference *default\_lnp\_prefix* is set for the originating peer, then the value of this preference is stored in *source\_lnp\_prefix* of the CDR.

Next, an LNP lookup is performed using the normalized called party number. If no number is found (using a longest match), no further manipulation is performed.

If an LNP number entry is found, and the *Routing Number* is set, the called party number is replaced by the routing number. Also, if the *Authoritative* flag is set in the corresponding LNP Carrier, and the called party number is not assigned to a local subscriber, the call is rejected. This ensures that numbers allocated to the system but not assigned to subscribers are dropped instead of routed to a peer.

---

#### **Important**



If the system is serving a local subscriber with only the routing number assigned (but not e.g. the premium number mapping to this routing number), the subscriber will not be found and the call will either be rejected if the called party premium number is within an authoritative carrier, or the call will be routed to a peer. This is due to the fact that the subscriber lookup is performed with the dialled number, but not the routing number fetched during LNP. So make sure to assign e.g. the premium number to the local subscriber (optionally in addition to the routing number if necessary using alias numbers) and do not use the LNP routing number mechanism for number mapping to local subscribers.

---

Next, if the the LNP carrier does not have the *Skip Rewriting* option set, the *LNP Rewrite Rules for Callee* are engaged. The rewrite rule set used is the one assigned to the originating peer or subscriber/domain via the *rewrite\_rule\_set* preference. The variables available in the match and replace part are, beside the standard variables for rewrite rules:

- `${callee_lnp_prefix}`: The prefix stored in the LNP Carrier
- `${callee_lnp_basenum}`: The actual number entry causing the match (may be shorter than the called party number due to longest match)

Typically, you would create a rewrite rule to prefix the called party number with the *callee\_lnp\_prefix* by matching `^([0-9]+)$` and replacing it by `${callee_lnp_prefix}\1`.

Once the LNP processing is completed, the system checks for further preferences to finalize the number manipulation. If the originating local subscriber or peer has the preference *lnp\_add\_npdi* set, the Request URI user-part is suffixed with `;npdi`. Next, if the preference *lnp\_to\_rn* is set, the Request URI user-part is suffixed with `;rn=LNP_ROUTING_NUMBER`, where *LNP\_ROUTING\_NUMBER* is the *Routing Number* stored for the number entry in the LNP database, and the originally called number is kept in place. For example, if *lnp\_to\_rn* is set and the number *1800123* is called, and this number has a routing number *1555123* in the LNP database, the resulting Request-URI is `sip:1800123;rn=1555123@example.org`.

Finally, the *destination\_lnp\_prefix* in the CDR table is populated either by the prefix defined in the Carrier of the LNP database if a match was found, or by the *default\_lnp\_prefix* preference of the destination peer or subscriber/domain.

#### 5.4.1.5 Blocking Calls Using LNP Data

The Sipwise NGCP provides means to allow or block calls towards ported numbers that are hosted by particular LNP carriers. Please visit Section 5.2.2.2 in the handbook to learn how this can be achieved.

#### 5.4.1.6 Transit Calls using LNP

If a call originated from a peer and the peer preference *force\_outbound\_calls\_to\_peer* is set to *force\_nonlocal\_lnp* (the *if callee is not local and is ported* selection in the panel), the call is routed back to a peer selected via the peering rules.

This ensures that if a number once belonged to your system and is ported out, but other carriers are still sending calls to you (e.g. selecting you as an anchor network), the affected calls can be routed to the carrier the number got ported to.

#### 5.4.1.7 CSV Format

The LNP database can be exported to CSV, and in the same format imported back to the system. On import, you can decide whether to drop existing data prior to applying the data from the CSV.

The CSV file format contains the fields in the following order:

```
carrier_name carrier_prefix number routing_number start end authoritative skip_rewrite
```

Table 2: LNP CSV Format

Name	Description
Carrier Name	The <i>Name</i> in the LNP Carriers table (string, e.g. <i>My Carrier</i> )
Carrier Prefix	The <i>Prefix</i> in the LNP Carriers table (string, e.g. <i>DD55</i> )
Number	The <i>Number</i> in the LNP Numbers table (E164 number, e.g. <i>1800666</i> )
Routing Number	The <i>Routing Number</i> in the LNP Numbers table (E164 number or empty, e.g. <i>1555666</i> )



Table 2: (continued)

Start	The <i>Start</i> in the LNP Numbers table (YYYY-MM-DD or empty, e.g. <i>2016-01-01</i> )
End	The <i>End</i> in the LNP Numbers table (YYYY-MM-DD or empty, e.g. <i>2016-12-30</i> )
Authoritative	The <i>Authoritative</i> flag in the LNP Carriers table (0 or 1)
Skip Rewrite	The <i>Skip Rewrite</i> flag in the LNP Carriers table (0 or 1)
Type	The <i>Type</i> tag in the LNP Numbers table (alphanumeric string, e.g. <i>mobile</i> )

## 5.4.2 External LNP via LNP API

External LNP relies on the *Sipwise LNP Daemon (lnpd)* which kamailio-proxy is talking to via a defined JSONRPC protocol. The proxy sends the A and B number to *lnpd*, which in the current release translates it to a SIP Message sent to an external server (typically a Squire SIP-to-INAP gateway. This external gateway is performing an SS7 INAP request to fetch the LNP result, which is passed back as a binary blob in a 3xx response to the *lnpd*. The *lnpd* extracts the TCAP body of the response and returns the information back to the proxy.

### 5.4.2.1 Enabling LNP lookup via API

In order to activate LNP lookup via API during call routing, the feature must be activated in `/etc/ngcp-config/config.yml`. Set these parameters:

- `kamailio→proxy→lnp→enabled: yes`
- `kamailio→proxy→lnp→type: api`
- `lnpd→enabled: yes`

There is a possibility to explicitly allow (whitelist) or deny (blacklist) certain number ranges for which an LNP lookup may be done. The relevant configuration parameters are at `kamailio→proxy→lnp→lnp_request_whitelist` and `kamailio→proxy→lnp→lnp_request_blacklist`. For each entry in the list a POSIX regex expression may be used, see the following example:

```
lnp:
  lnp_request_whitelist:
    - '^9'
    - '^800'
  lnp_request_blacklist:
    - '^1'
    - '^900'
    - '^110'
    - '^112'
```

Interpretation of the above lists (that are based on numbers represented in national format):

- **whitelist:** *do* LNP lookup for any called number that starts with *9* or *800*
- **blacklist:** *do not* perform LNP lookup for any called number that starts with *1*, *900*, *110* or *112*



#### Important

If both whitelist and blacklist are defined, the LNP lookup is only performed when the called number matches any of the whitelist patterns and does not match any of the blacklist patterns.

---

### 5.4.2.2 The Redundancy Feature

It is possible to set up *LNP daemon* to provide a kind of redundant service to the Proxy. This means the *LNP daemon* will send its LNP query to more LNP serving nodes that are predefined in a list. (See [Configuration of LNP daemon](#) Section 5.4.2.3 chapter for details.) The LNP query may happen in 2 ways:

- **round-robin:** *LNP daemon* sends the query to one of the serving nodes then waits for the response for a configurable timeout. If it does not get the response in time, it sends the LNP query to the next serving node.
- **parallel:** *LNP daemon* sends the query to all of the serving nodes then waits for the response, and will accept the first response that it receives.

### 5.4.2.3 Configuration of Sipwise LNP Daemon

*LNP daemon* takes its active configuration from `/etc/ngcp-lnpd/config.yml` file. The file is generated automatically—when a new NGCP configuration is applied (`ngcpcfg apply...`)—from the main Sipwise NGCP configuration file: `/etc/ngcp-config/config.yml` and a **template:** `/etc/ngcp-config/template/etc/ngcp-lnpd/config.yml.tt2`. System administrators are only expected to modify the `lnpd.config` section of main configuration file `/etc/ngcp-config/config.yml`.

A sample *LNP daemon* configuration file (`/etc/ngcp-lnpd/config.yml`) looks like:

```
daemon:
  json-rpc:
    ports:
      - 54321
      - 12345
    interfaces:
      - 127.0.0.1
      - 192.168.1.90
      - ::1

  sip:
    port: 5095
    address: 0.0.0.0
```

```
threads: 4
foreground: false
pidfile: /tmp/lmpd.pid
loglevel: 7

instances:
  default:
    module: sigtran
    destination: 192.168.1.99
    from-domain: test.example.com
    headers:
      - header: INAP-Service-Key
        value: 2
    reply:
      tcap: raw-tcap
  redundant:
    module: sigtran
    destinations:
      - 192.168.1.99
      - 192.168.1.95
      - 192.168.1.90
    mechanism: round-robin
    retry-time: 30
    timeout: 5
    from-domain: test.example.com
    headers:
      - header: INAP-Service-Key
        value: 2
    reply:
      tcap: raw-tcap
  parallel:
    module: sigtran
    destinations:
      - 192.168.1.99
      - 192.168.1.95
      - 192.168.1.90
    mechanism: parallel
    retry-time: 30
    timeout: 10
    from-domain: test.example.com
    headers:
      - header: INAP-Service-Key
        value: 2
    reply:
      tcap: raw-tcap
  mock1:
    module: mock-tcap
```

```

numbers:
  - number: '4311003'
    routing-number: '4318881003'
reply:
  tcap: raw-tcap

```

The corresponding NGCP main configuration file contains:

```

daemon:
  foreground: 'false'
  json-rpc:
    ports:
      - '54321'
      - '12345'
  loglevel: '7'
  sip:
    port: '5095'
    threads: '4'
  instances:
<< These are the same entries as in /etc/ngcp-lnpd/config.yml file >>

```

### Description of configuration parameters in `/etc/ngcp-config/config.yml` file

- **daemon section:**
  - `foreground`: determines if the LNP daemon runs as foreground or background process
  - `json-rpc.ports`: port numbers where LNP daemon listens for incoming JSONRPC requests from NGCP Proxy
  - `loglevel`: how detailed information LNP daemon writes in its log file
  - `sip.port`: listening port number used for SIP sessions with LNP serving nodes; LNP daemon will listen on first available (shared) IP address that is taken from `/etc/ngcp-config/network.yml` file
  - `threads`: number of threads LNP daemon will use internally; this value determines how many requests the daemon can serve in parallel
- **instances section:** at least one `default` instance must be defined here. Others are also useful for providing redundancy, please check `redundant` and `parallel` entries above.
  - `module`: only `sigtran` is used for normal operations



#### Important

The module `mock-tcap` is only meant for developers. In this case the LNP daemon does not produce a SIP request that it sends to LNP serving nodes, but instead it uses the `numbers` parameter to match a called number with a routing number. The `numbers` parameter contains a list of `number` — `routing-number` pairs and is used as a database for number lookups. Finally LNP daemon returns the routing number as a response on LNP query.

- `destinations`: list of nodes to which LNP daemon sends the LNP query
- `mechanism`: either `parallel` or `round-robin`, defining the method of redundant queries

- `retry-time`: a period of time in seconds while LNP daemon considers an LNP serving node being unreachable after an LNP query timeout
- `timeout`: the period of time while LNP daemon waits for a response on an LNP query from one of the LNP serving nodes  
*PLEASE NOTE*: `retry-time` and `timeout` are used with both the parallel and the round-robin redundancy methods
- `from-domain`: the domain that will be used in SIP *From* header when LNP daemon sends the LNP query
- `headers`: this is a list of header name—value pairs; these custom headers will be included in SIP request that LNP daemon sends to an LNP serving node
- `reply.tcap`: determines the format of reply sent to NGCP Proxy; currently only `raw-tcap` is supported, which means LNP daemon will not decode the TCAP response it gets from an LNP serving node but it forwards the raw TCAP message body

## 5.5 Emergency Mapping

As opposed to the [Simple Emergency Number Handling](#) Section 4.7.5.1 solution, the Sipwise NGCP supports an advanced emergency call handling method, called *emergency mapping*. The main idea is: instead of obtaining a statically assigned emergency prefix / suffix from subscriber preferences, NGCP retrieves an emergency routing prefix from a central emergency call routing table, according to the current location of the calling subscriber.

The following figure shows the overview of emergency call processing when using *emergency mapping* feature:

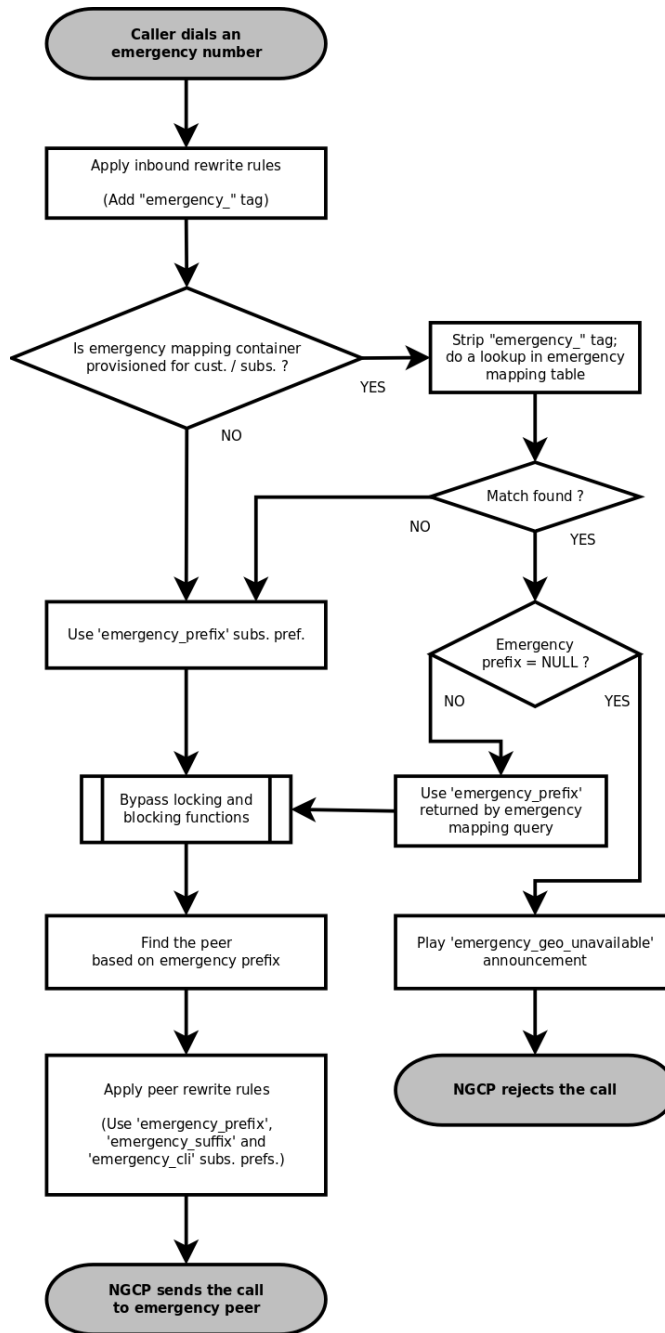


Figure 36: Emergency Call Handling with Mapping

### 5.5.1 Emergency Mapping Description

Emergency numbers per geographic location are mapped to different routing prefixes not deriveable from an area code or the emergency number itself. This is why a **global emergency mapping table** related to resellers is introduced, allowing to map emergency numbers to their geographically dependent routing numbers.

The geographic location is referenced by a location ID, which has to be populated by a north-bound provisioning system. No towns, areas or similar location data is stored on the NGCP platform. The locations are called *Emergency Containers* on NGCP.

The actual emergency number mapping is done per location (per *Emergency Container*), using the so-called *Emergency Mapping* entries. An *Emergency Mapping* entry assigns a routing prefix, valid only in a geographic area, to a generic emergency number (for example *112* in Europe, *911* in the U.S.A.) or a country specific one (for example *133*).

---

**Note**

As of mr4.5 version, the NGCP performs an exact match on the emergency number in the emergency routing table.

---

*Emergency Containers* may be assigned to various levels of the client hierarchy within NGCP. The following list shows such levels with each level overriding the settings of the previous one:

1. Customer or Domain
2. Customer Location, which is a territory representing a subset of the customer's subscribers, defined as one or more IP subnets.
3. Subscriber

---

**Note**

Please be aware that *Customer Location* is not necessarily identical to the "location" identified through an *Emergency Container*.

---

Once the emergency routing prefix has been retrieved from the emergency mapping table, call processing continues in the same way as in case of simple emergency call handling.

### 5.5.2 Emergency Mapping Configuration

The administrative web panel of NGCP provides the configuration interface for emergency mapping. Please navigate to *Settings* → *Emergency Mapping* menu item first, in order to start configuring the mapping.

An *Emergency Container* must be created, before the mapping entries can be defined. Press *Create Emergency Container* to start this. An example of a container is shown here:

Edit Emergency Containers
✕

Reseller

Search:

#	Name	Contract #	Status	
1	default	1	active	<input checked="" type="checkbox"/>
3	api_test test reseller	137	active	<input type="checkbox"/>
5	patched name 1494894408	179	active	<input type="checkbox"/>
7	test reseller 1494894408 2	181	active	<input type="checkbox"/>
9	test reseller 1494894408 3	183	active	<input type="checkbox"/>

Showing 1 to 4 of 8 entries

⏪
⏩
1
2
⏪
⏩

Create Reseller

Name

EmergCont\_1

Save

Figure 37: Creating an Emergency Container

You have to select a `Reseller` that this container belongs to, and enter a `Name` for the container, which is an arbitrary text.

---

**Tip**

The platform administrator has to create as many containers as the number of different geographic areas (locations) the subscribers are expected to be in.

---

As the second step of emergency mapping provisioning, the *Emergency Mapping* entries must be created. Press *Create Emergency Mapping* to start this step. An example is shown here:



Emergency Mapping Container

Search:

#	Reseller	Name	
1	default	EmergCont_1	<input type="checkbox"/>
3	default	EmergCont_2	<input type="checkbox"/>

Showing 1 to 2 of 2 entries

Code

Prefix

Figure 38: Creating an Emergency Mapping Entry

The following parameters must be set:

- **Container:** select an emergency mapping container (i.e. a location ID)
- **Code:** the emergency number that subscribers will dial
- **Prefix:** the routing prefix that belongs to the particular emergency service within the selected location

Once all the necessary emergency mappings have been defined, the platform administrator will see a list of containers and mapping entries:

### Emergency Mappings

← Back
★ Download CSV
★ Upload CSV

#### Emergency Containers

★ Create Emergency Container

Show 5 entries Search:

#	Reseller	Name
1	default	EmergCont_1
3	default	EmergCont_2

Showing 1 to 2 of 2 entries

#### Emergency Mappings

★ Create Emergency Mapping

Show 5 entries Search:

#	Container	Reseller	Emergency Number	Emergency Prefix
1	EmergCont_1	default	133	E1_133_
3	EmergCont_1	default	144	E1_144_
5	EmergCont_2	default	133	E2_133_

Figure 39: Emergency Mapping List

The emergency number mapping is now defined. As the next step, the platform administrator has to assign the emergency containers to *Customers / Domains / Customer Locations* or *Subscribers*. We'll take an example with a *Customer*: select the customer, then navigate to *Details* → *Preferences* → *Number Manipulations*. In order to assign a container, press the *Edit* button and then select one container from the drop-down list:

Customer #205 - Preferences

← Back Expand Groups

Call Blockings

Access Restrictions

**Number Manipulations**

Attribute	Name	Value	
emergency_prefix	Emergency Prefix variable		
emergency_suffix	Emergency Suffix variable		
emergency_cli	Emergency CLI		
emergency_mapping_container	Emergency Mapping Container	EmergCont_2	Edit

Internals

Figure 40: Assigning an Emergency Mapping Container

### Rewrite Rules for Emergency Mapping

Once emergency containers and emergency mapping entries are defined, the NGCP administrator has to ensure that the proper number manipulation takes place, before initiating any emergency call towards peers.



#### Important

Please don't forget to define the rewrite rules for peers—particularly: *Outbound Rewrite Rules for Callee*—as described in [Normalize Emergency Calls for Peers](#) Section 4.7.5.3 section of the handbook.

#### 5.5.2.1 Emergency Calls Not Allowed

There is a special case when the dialed number is recognized as an emergency number, but the emergency number is not available for the geographic area the calling party is located in.

In such a case the emergency mapping lookup will return an emergency prefix, but the value of this will be NULL. Therefore the call is rejected and an announcement is played. The announcement is a newly defined sound file referred as `emergency_geo_unavailable`.

It is possible to configure the rejection code and reason in `/etc/ngcp-config/config.yml` file, the parameters are: `kamailio.proxy.early_rejects.emergency_invalid.announce_code` and `kamailio.proxy.early_rejects.emergency_invalid.announce_reason`.

### 5.5.2.2 Bulk Upload or Download of Emergency Mapping Entries

The Sipwise NGCP offers the possibility to upload / download emergency mapping entries in form of CSV files. This operation is available for each reseller, and is very useful if a reseller has many mapping entries.

#### **Downloading Emergency Mapping List**

One has to navigate to *Settings* → *Emergency Mapping* menu and then press the *Download CSV* button to get the list of mapping entries in a CSV file. First the reseller must be selected, then the *Download* button must be pressed. As an example, the entries shown in "Emergency Mapping List" picture above would be written in the file like here below:

```
EmergCont_1,133,E1_133_  
EmergCont_1,144,E1_144_  
EmergCont_2,133,E2_133_
```

The **CSV file** has a plain text format, each line representing a mapping entry, and contains the following **fields**:

- Container name, as defined in *Emergency Containers*
- Emergency Number
- Emergency Prefix

#### **Uploading Emergency Mapping List**

Uploading a CSV file with emergency mapping entries may be started after pressing the *Upload CSV* button. The following data must be provided:

- *Reseller*: selected from the list
- *Upload mapping*: the CSV file must be selected after pressing the *Choose File* button
- *Purge existing*: an option to purge existing emergency mapping entries that belong to the selected reseller, before populating the new mapping data from the file

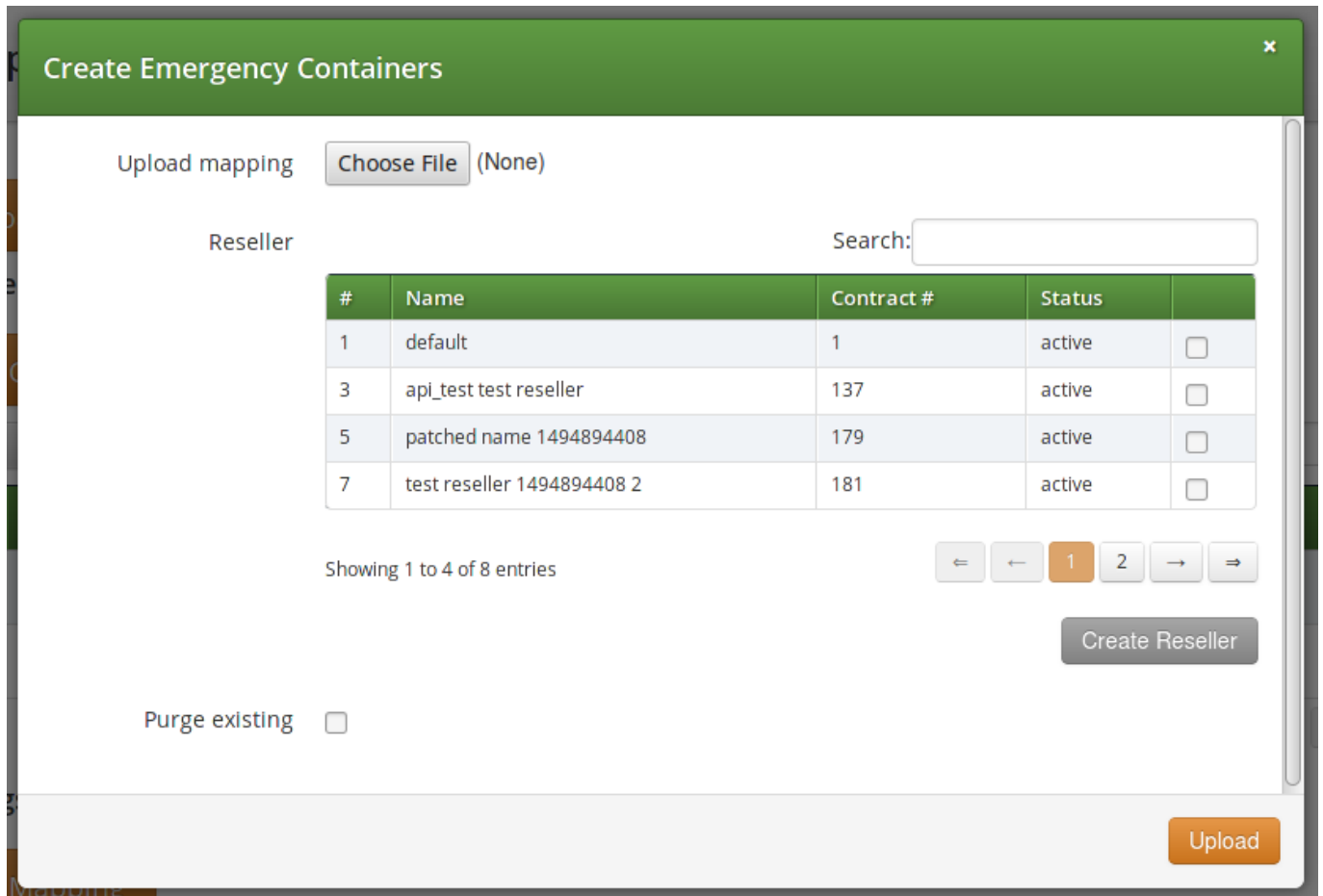


Figure 41: Uploading Emergency Mapping Data

The CSV file for the upload has the same format as the one used for download.

## 5.6 Emergency Priorization

The NGCP can potentially host *privileged subscribers* that offer emergency or at least prioritized services (civil defence, police etc.). In case of an emergency, the platform has to be free'd from any SIP flows (calls, registrations, presence events etc.) which do not involve those privileged subscribers.

Such an exceptional condition is called **emergency mode** and it can be activated for all domains on the system, or only for selected domains.

Once emergency mode is activated, Sipwise NGCP will immediately apply the following restrictions on new SIP requests or existing calls:

- Any SIP requests (calls, registrations etc.) from subscribers within the affected domains, who are not marked as privileged, are rejected.
- Any calls from peers not targeting privileged subscribers are rejected.

- Any active calls which do not have a privileged subscriber involved are terminated.

Calls from non-privileged subscribers to emergency numbers are still allowed.

### 5.6.1 Call-Flow with Emergency Mode Enabled

Typical call-flows of emergency mode will be shown in this section of the handbook. We have the following assumptions:

- Emergency prioritization has been enabled on system-level
- There is a domain for which the emergency mode has been activated
- There is a privileged subscriber in that domain
- A generic peering connection has been configured for non-emergency calls
- A dedicated peering connection has been configured for emergency calls

The examples do not show details of SIP messages, but rather give a high-level overview of the call-flows.

1. A **non-privileged** subscriber makes a call **to another non-privileged subscriber**. Result: the call will be **rejected**.

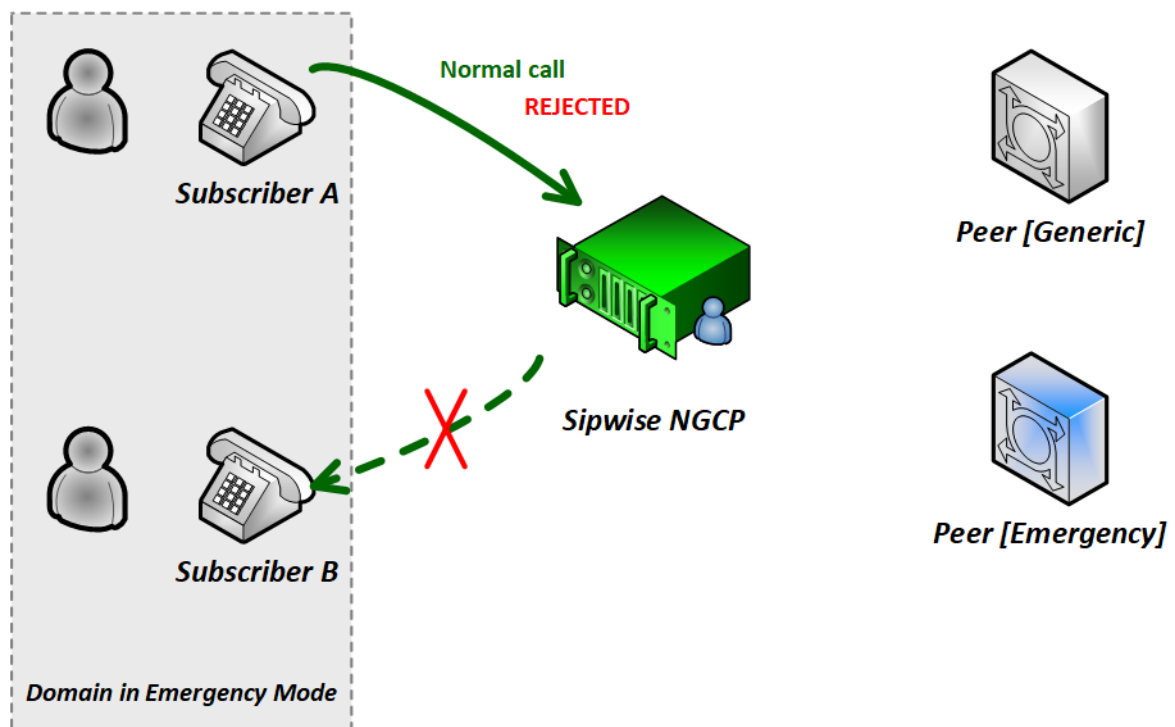


Figure 42: Call-flow in Emergency Mode 1. (Std to Std)

2. A **non-privileged** subscriber makes a call **to an external subscriber (via peer)**. Result: the call will be **rejected**.

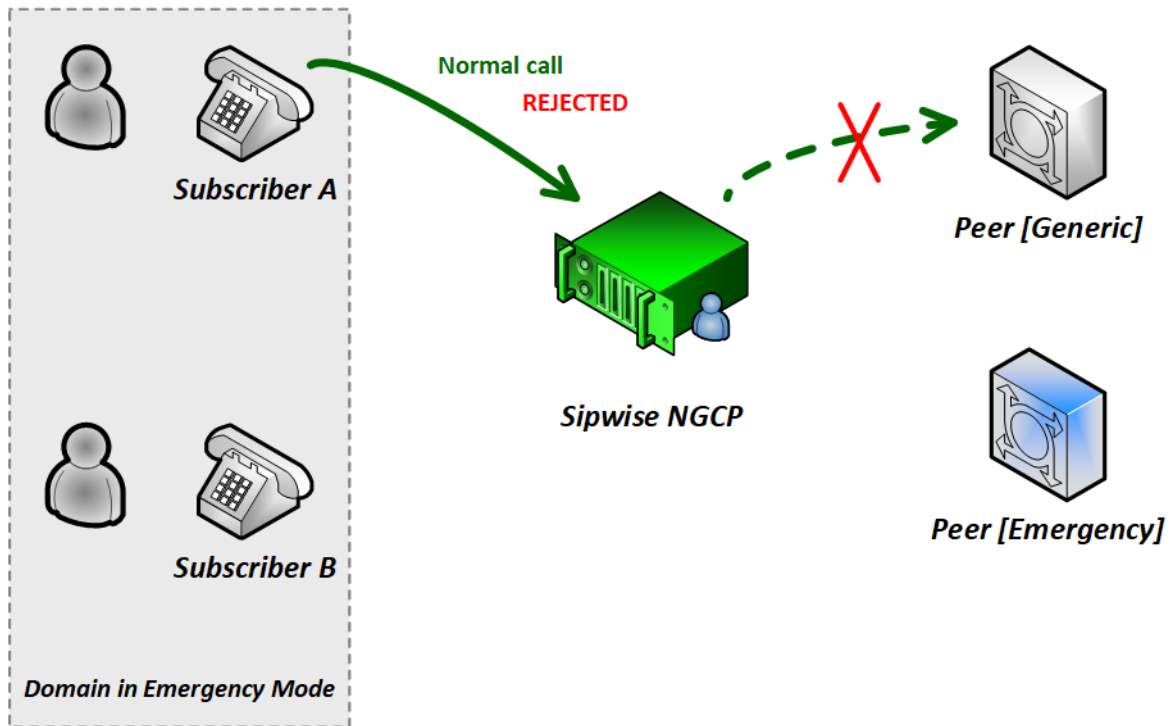


Figure 43: Call-flow in Emergency Mode 2. (Std to Peer)

3. A **non-privileged** subscriber makes a call **to a privileged subscriber**. Result: the call will be **accepted**.

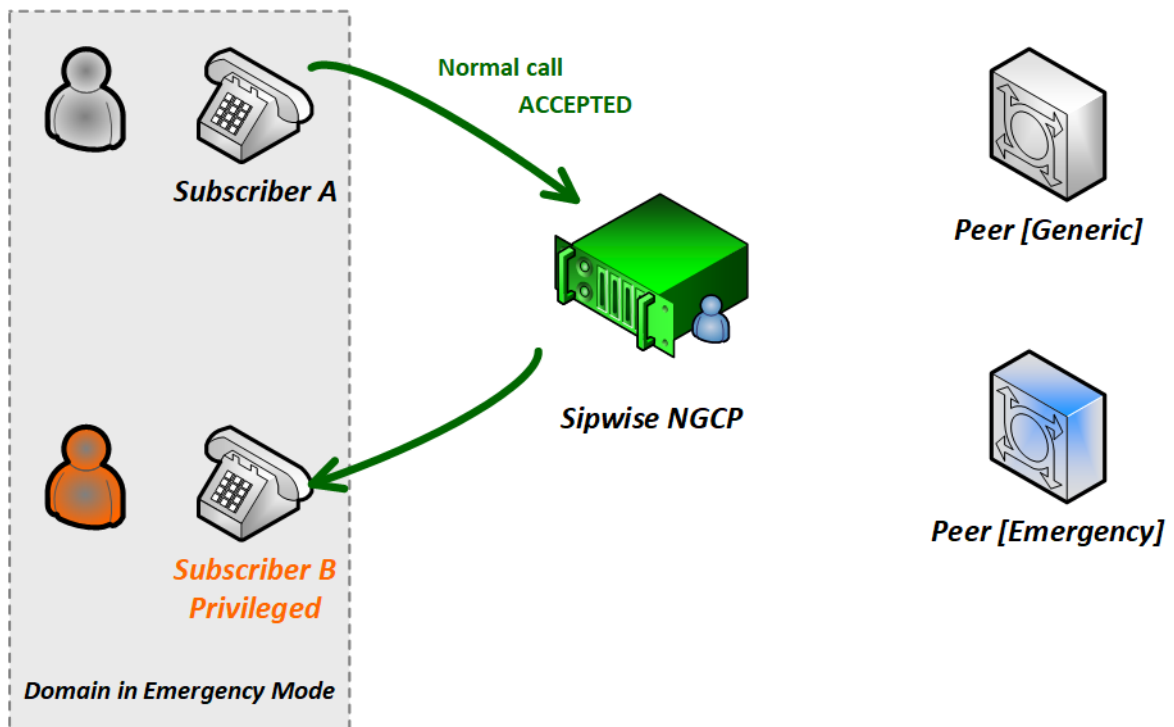


Figure 44: Call-flow in Emergency Mode 3. (Std to Priv)

4. A **non-privileged** subscriber makes a call to an emergency number. Result: the call will be **accepted**.

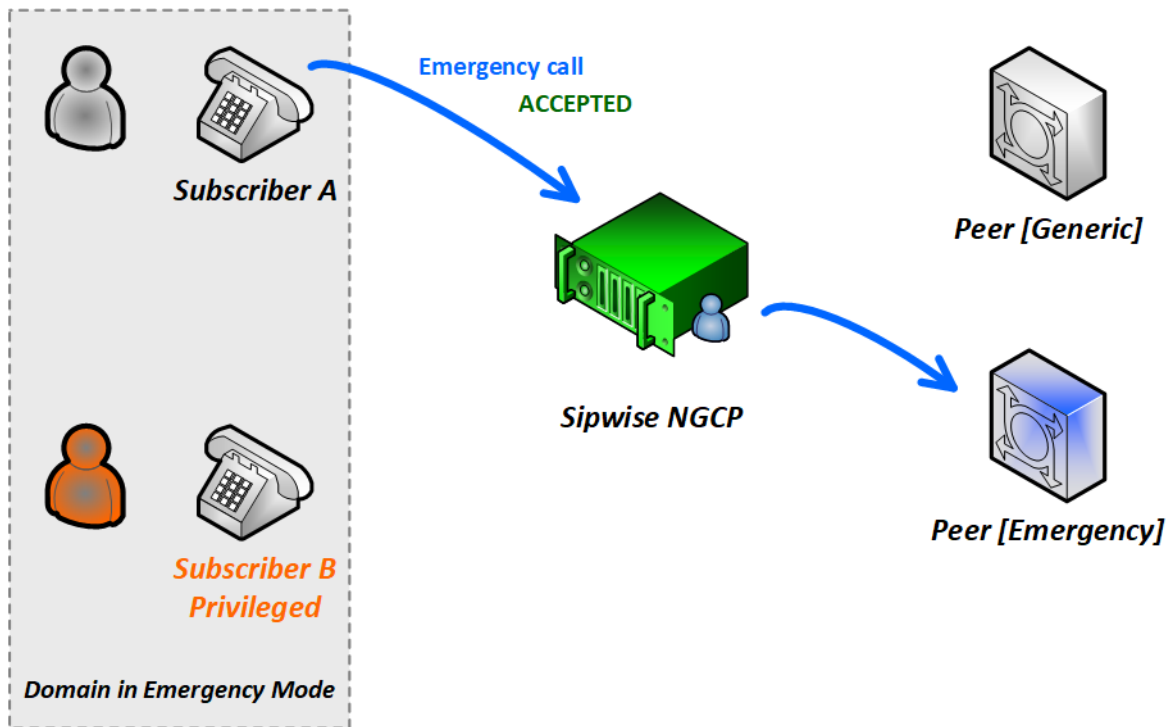


Figure 45: Call-flow in Emergency Mode 4. (Std to Emerg)

5. A **privileged** subscriber makes a call to a non-privileged subscriber. Result: the call will be **accepted**.

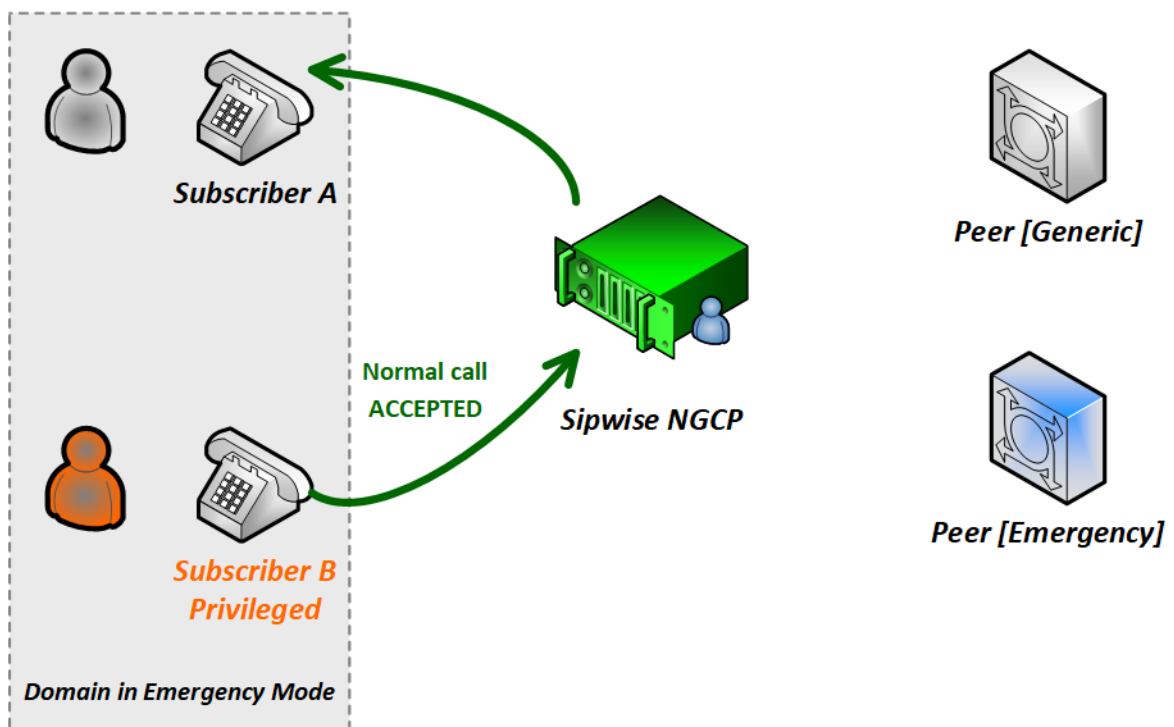


Figure 46: Call-flow in Emergency Mode 5. (Priv to Std)



6. A **privileged** subscriber makes a call **to an external subscriber (via peer)**. Result: the call will be **accepted**.

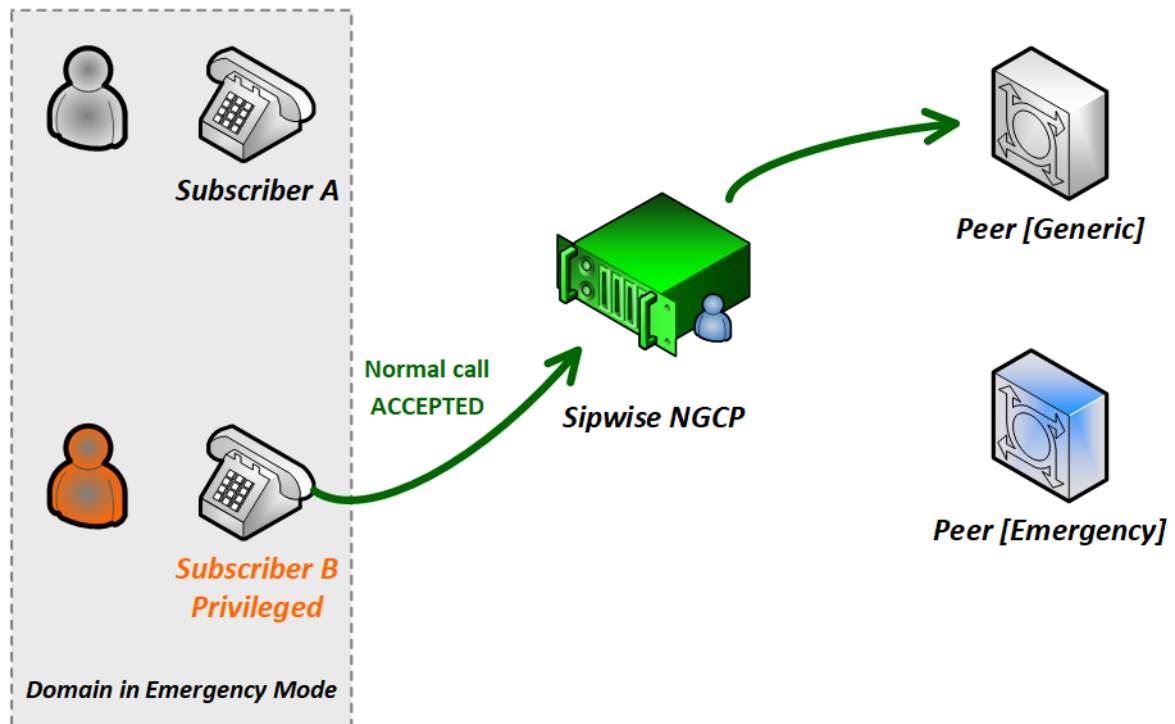


Figure 47: Call-flow in Emergency Mode 6. (Priv To Peer)

## 5.6.2 Configuration of Emergency Mode

The platform operator has to perform 2 steps of configuration so that the emergency mode can be activated. After the configuration is completed it is necessary to explicitly activate emergency mode, which can be accomplished as described in Section 5.6.3 later.

### 1. System-level Configuration

The emergency prioritization function must be enabled for the whole system, otherwise emergency mode can not be activated. The platform operator has to set `kamailio.proxy.emergency_priorization.enabled` configuration parameter value to "yes" in the main configuration file `/etc/ngcp-config/config.yml`. Afterwards changes have to be applied in the usual way, with the command: `ngcpcfg apply "Enabled emergency prioritization"`

In order to learn about other parameters related to emergency prioritization please refer to Section B.1.16 part of the handbook.

### 2. Subscriber-level Configuration

The platform operator (or any administrator user) has the capability to declare a subscriber privileged, so that the subscriber can initiate and receive calls when emergency mode has been activated on the NGCP. In order to do that the administrator has to navigate to *Settings* → *Subscribers* → *select the subscriber* → *Details* → *Preferences* → *Internals* → *emergency\_priorization* on the **administrative web interface**, and press the *Edit* button.

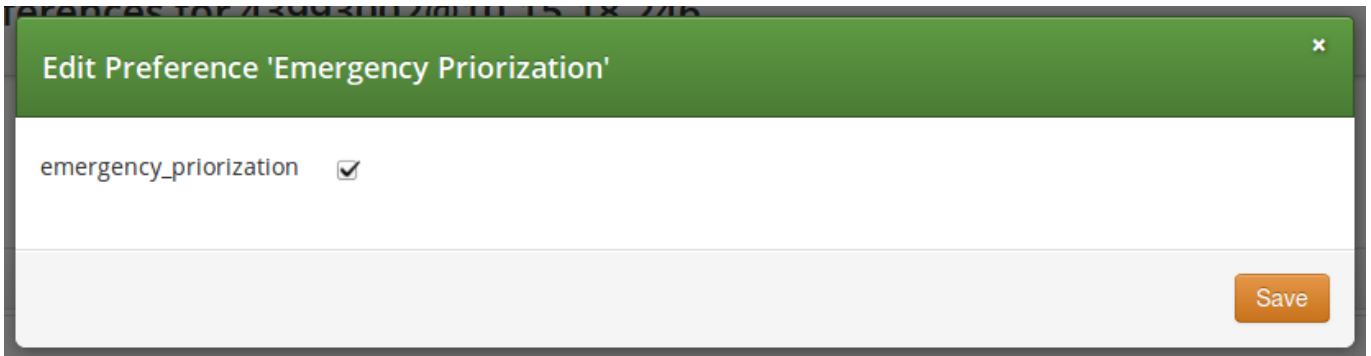


Figure 48: Emergency Priorization of Subscriber

The checkbox `emergency_priorization` has to be ticked and then press the `Save` button.

The same privilege can be added via the **REST API** for a subscriber: a HTTP PUT/PATCH request must be sent on `/api/subscriberpreferences/id` resource and the `emergency_priorization` property must be set to `"true"`.

### 5.6.3 Activating Emergency Mode

The platform operator can activate emergency mode for a single or multiple domains in 3 different ways:

- via the administrative web interface
- via the REST API
- via a command-line tool



#### Important

The interruption of ongoing calls is only possible with the command-line tool! Activating emergency mode for domains via the web interface or REST API will only affect upcoming calls.

---

**1. Activate emergency mode via web interface:** this way of activation is more appropriate if only a single (or just a few) domain is affected. Please navigate to `Settings` → `Domains` → `select a domain` → `Preferences` → `Internals` → `emergency_mode_enabled` → `Edit`.

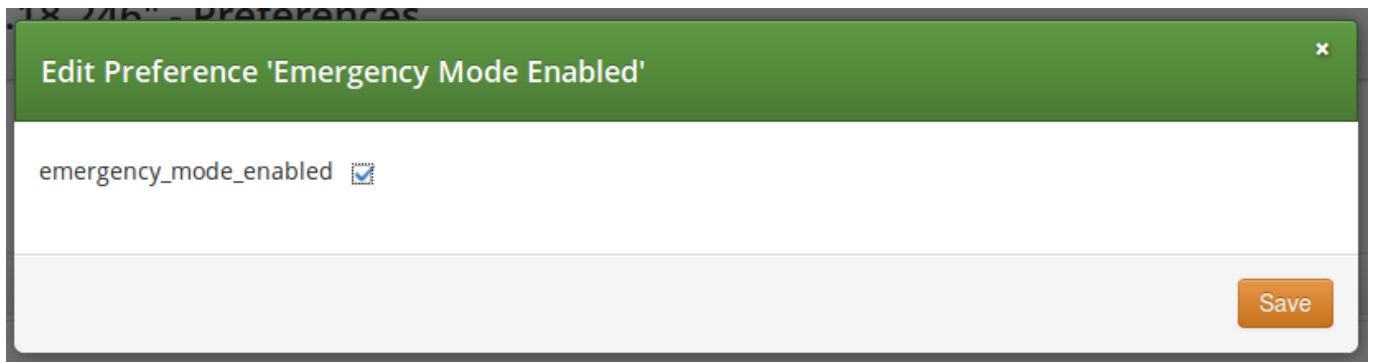


Figure 49: Activate Emergency Mode of Domain

The checkbox `emergency_mode_enabled` has to be ticked and then press the `Save` button.

**2. Activate emergency mode via REST API:** this way of activation is more appropriate if only a single (or just a few) domain is affected.

For that purpose a HTTP PUT/PATCH request must be sent on `/api/domainpreferences/id` resource and the `emergency_mode_enabled` property must be set to `"true"`.

**3. Activate emergency mode using a command-line tool:** the Sipwise NGCP provides a built-in script that may be used to enable/disable emergency mode for some particular or all domains.

- Enable emergency mode:

```
> ngcp-emergency-mode enable <all|[domain1 domain2 ...]>
```

- Disable emergency mode:

```
> ngcp-emergency-mode disable <all|[domain1 domain2 ...]>
```

- Query the status of emergency mode:

```
> ngcp-emergency-mode status <all|[domain1 domain2 ...]>
```

## 5.7 Header Manipulation

### 5.7.1 Header Filtering

Adding additional SIP headers to the initial INVITEs relayed to the callee (second leg) is possible by modifying the following template file: `/etc/ngcp-config/templates/etc/ngcp-sems/etc/ngcp.sbcprofile.conf.customtt.tt2`. The following section can be changed:

```
header_filter=whitelist
header_list=[%IF kamilio.proxy.debug == "yes"%]P-NGCP-CFGTEST, [%END%]
P-R-Uri, P-D-Uri, P-Preferred-Identity, P-Asserted-Identity, Diversion, Privacy,
```

```
Allow, Supported, Require, RAck, RSeq, Rseq, User-Agent, History-Info, Call-Info
[%IF kamailio.proxy.presence.enable == "yes"%], Event, Expires,
Subscription-State, Accept[%END%] [%IF kamailio.proxy.allow_refer_method
== "yes"%], Referred-By, Refer-To, Replaces[%END%]
```

By default the system will remove from the second leg all the SIP headers which are not in the above list. If you want to keep some additional/custom SIP headers, coming from the first leg, into the second leg you just need to add them at the end of the *header\_list*= list. After that, as usual, you need to apply and push the changes. In this way the system will keep your headers in the INVITE sent to the destination subscriber/peer.



### Warning

DO NOT TOUCH the list if you don't know what you are doing.

## 5.7.2 Codec Filtering

Sometimes you may need to filter some audio CODEC from the SDP payload, for example if you want to force your subscribers to do not talk a certain codecs or force them to talk a particular one. To achieve that you just need to change the `/etc/ngcp-config/config.yml`, in the following section:

```
sdp_filter:
  codecs: PCMA,PCMU,telephone-event
  enable: yes
  mode: whitelist
```

In the example above, the system is removing all the audio CODECS from the initial INVITE except G711 alaw,ulaw and telephone-event. In this way the callee will be notified that the caller is able to talk only PCMA. Another example is the `blacklist` mode:

```
sdp_filter:
  codecs: G729,G722
  enable: yes
  mode: blacklist
```

In this way the G729 and G722 will be removed from the SDP payload. In order to apply the changes, run

```
ngcpcfg apply 'Enable CODEC filtering'
ngcpcfg push all
```

## 5.7.3 Enable History and Diversion Headers

It may be useful and mandatory - specially with NGN interconnection - to enable SIP History header and/or Diversion header for outbound requests to a peer or even for on-net calls. In order to do so, you should enable the following preferences in Domain's and Peer's Preferences:

- Domain's Preferences: *inbound\_uprn* = **Forwarder's NPN**
- Peer's Preferences: *outbound\_history\_info* = **UPRN**
- Peer's Preferences: *outbound\_diversion* = **UPRN**
- Domain's Preferences: *outbound\_history\_info* = **UPRN** (if you want to allow History Header for on-net call as well)
- Domain's Preferences: *outbound\_diversion* = **UPRN** (if you want to allow Diversion Header for on-net call as well)

## 5.8 SIP Trunking with SIPconnect

### 5.8.1 User provisioning

For the purpose of external SIP-PBX interconnect with sip:carrier the platform admin should create a subscriber with multiple aliases representing the numbers and number ranges served by the SIP-PBX.

- Subscriber username - any SIP username that forms an "email-style" SIP URI.
- Subscriber Aliases - numbers in the global E.164 format without leading plus.

To configure the Subscriber, go to *Settings*→*Subscribers* and click *Details* on the row of your subscriber. There, click on the *Preferences* button on top.

You should look into the *Number Manipulations* and *Access Restrictions* sections in particular, which control the calling and called number presentation.

### 5.8.2 Inbound calls routing

Enable preference *Number Manipulations*→*e164\_to\_ruri* for routing inbound calls to SIP-PBX. This ensures that the Request-URI will comprise a SIP-URI containing the dialed alias-number as user-part, instead of the user-part of the registered AOR (which is normally a static value).

### 5.8.3 Number manipulations

The following sections describe the recommended configuration for correct call routing and CLI presentation according to the SIPconnect 1.1 recommendation.

#### 5.8.3.1 Rewrite rules

The SIP PBX by default inherits the domain dialplan which usually has rewrite rules applied to normal Class 5 subscribers with inbound rewrite rules normalizing the dialed number to the E.164 standard. If most users of this domain are Class 5 subscribers the dialplan may supply calling number in national format - see Section 4.7. While the SIP-PBX trunk configuration can be sometimes amended it is a good idea in sense of SIPconnect recommendation to send only the global E.164 numbers.

Moreover, in mixed environments with the sip:carrier Cloud PBX sharing the same domain with SIP trunking (SIP-PBX) customers the subscribers may have different rewrite rules sets assigned to them. The difference is caused by the fact that the dialplan for Cloud PBX is fundamentally different from the dialplan for SIP trunks due to extension dialing, where the Cloud PBX subscribers use the break-out code (see Section 16.1.2) to dial numbers outside of this PBX.

The SIPconnect compliant numbering plan can be accommodated by assigning Rewrite Rules Set to the SIP-PBX subscriber. Below is a sample Rewrite Rule Set for using the global E.164 numbers with plus required for the calling and called number format compliant to the recommendation.

#### INBOUND REWRITE RULE FOR CALLER

- Match Pattern: `^(00|\+)([1-9][0-9]+)$`
- Replacement Pattern: `\2`
- Description: International to E.164
- Direction: Inbound
- Field: Caller

#### INBOUND REWRITE RULE FOR CALLEE

- Match Pattern: `^(00|\+)([1-9][0-9]+)$`
- Replacement Pattern: `\2`
- Description: International to E.164
- Direction: Inbound
- Field: Callee

#### OUTBOUND REWRITE RULE FOR CALLER

- Match Pattern: `^([1-9][0-9]+)$`
- Replacement Pattern: `+\1`
- Description: For the calls to SIP-PBX add plus to E.164
- Direction: Outbound
- Field: Caller

#### OUTBOUND REWRITE RULE FOR CALLEE

- Match Pattern: `^([1-9][0-9]+)$`
- Replacement Pattern: `+\1`
- Description: For the calls to SIP-PBX add plus to E.164

- Direction: Outbound
- Field: Callee

Assign the aforementioned Rewrite Rule Set to the SIP-PBX subscribers.



#### Warning

Outbound Rewrite Rules for Callee shall NOT be applied to the calls to normal SIP UAs like IP phones since the number with plus does not correspond to their SIP username.

---

### 5.8.3.2 User parameter

The following configuration is needed for your platform to populate the From and To headers and Request-URI of the INVITE request with "user=phone" parameter as per RFC 3261 Section 19.1.1 (if the user part of the URI contains telephone number formatted as a telephone-subscriber).

- Domain's Preferences: *outbound\_from\_user\_is\_phone* = Y
- Domain's Preferences: *outbound\_to\_user\_is\_phone* = Y

### 5.8.3.3 Forwarding number

The following is our common configuration that covers the calling number presentation in a variety of use-cases, including the incoming calls, on-net calls and Call Forward by the platform:

- Domain's Preferences: *inbound\_uprn* = **Forwarder's NPN**
- Domain's Preferences: *outbound\_from\_user* = **UPRN (if set) or User-Provided Number**
- Domain's Preferences: *outbound\_pai\_user* = **UPRN (if set) or Network-Provided Number**
- Domain's Preferences: *outbound\_history\_info* = **UPRN** (if the called user expects History-Info header)
- Domain's Preferences: *outbound\_diversion* = **UPRN** (if the called user expects Diversion header)
- Domain's Preferences: *outbound\_to\_user* = **Original (Forwarding) called user** if the callee expects the number of the subscriber forwarding the call, otherwise leave default.

The above parameters can be tuned to operator specifics as required. You can of course override these settings in the Subscriber Preferences if particular subscribers need special settings.

---

#### Tip

On outgoing call from SIP-PBX subscriber the Network-Provided Number (NPN) is set to the *cli* preference prefilled with main E.164 number. In order to have the full alias number as NPN on outgoing call set preference *extension\_in\_npn* = Y.

---

**Externally forwarded call** If the call forward takes place inside the SIP-PBX it can use one of the following specification for signaling the diversion number to the platform:

- using **Diversion** method (RFC 5806): configure Subscriber's Preferences: *inbound\_uprn* = **Forwarder's NPN / Received Diversion**
- using **History-Info** method (RFC 7044): NGCP platform extends the History-Info header received from the PBX by adding another level of indexing according to the specification RFC 7044.

#### 5.8.3.4 Allowed CLIs

- For correct calling number presentation on outgoing calls, you should include the pattern matching all the alias numbers of SIP-PBX or each individual alias number under the *allowed\_clis* preference.
- If the signalling calling number (usually taken from From user-part, see *inbound\_upn* preferences) does not match the *allowed\_clis* pattern, the *user\_cli* or *cli* preference (Network-Provided Number) will be used for calling number presentation.

#### 5.8.4 Registration

SIP-PBX can use either Static or Registration Mode. While SIPconnect 1.1 continues to require TLS support at MUST strength, one should note that using TLS for signaling does not require the use of the SIPS URI scheme. SIPS URI scheme is obsolete for this purpose.

**Static Mode** While SIPconnect 1.1 allows the use of Static mode, this poses additional maintenance overhead on the operator. The administrator should create a static registration for the SIP-PBX: go to Subscribers, *Details*→*Registered Devices*→*Create Permanent Registration* and put address of the SIP-PBX in the following format: sip:username@ipaddress:5060 where username=username portion of SIP URI and ipaddress = IP address of the device.

**Registration Mode** It is recommended to use the Registration mode with SIP credentials defined for the SIP-PBX subscriber.



#### Important

The use of RFC 6140 style "bulk number registration" is discouraged. The SIP-PBX should register one AOR with email-style SIP URI. The sip:carrier will take care of routing the aliases to the AOR with *e164\_to\_ruri* preference.

---

#### 5.8.4.1 Trusted Sources

If a SIP-PBX cannot perform the digest authentication, you can authenticate it by its source IP address in sip:carrier. To configure the IP-based authentication, go to the subscriber's preferences (*Details*→*Preferences*→*Trusted Sources*) and specify the IP address of the SIP-PBX in the *Source IP* field.

To authenticate multiple subscribers from the same IP address, use the *From* field to distinguish these subscribers.

When this feature is configured for a subscriber, the sip:carrier authenticates all calls that arrive from the specified IP address without challenging them.



**Important**

If the same IP address and the FROM field are mistakenly specified as trusted for different subscribers, the sip:carrier will not know which subscriber to charge for the call and will randomly select one.

---

## 5.9 Trusted Subscribers

In some cases, when you have a device that cannot authenticate itself against sip:carrier, you may need to create a *Trusted Subscriber*. Trusted Subscribers use IP-based authentication and they have a Permanent SIP Registration URI in order to receive messages from sip:carrier.

In order to make a regular subscriber trusted, perform the following extra steps:

- Create a permanent registration via (*Subscribers*→*Details*→*Registered Devices*→*Create Permanent Registration*)
- Add the IP address of the device as Trusted Source in your subscriber's preferences (*Details*→*Preferences*→*Trusted Sources*).

This way, all SIP messages coming from the device IP will be considered trusted (and get authenticated just by the source IP). All the SIP messages forwarded to the devices will be sent to the SIP URI specified in the subscriber's permanent registration.

## 5.10 Peer Probing

The basic way of selecting the appropriate peering server, where an outbound call can be routed to, has already been described in Section 4.6.2.3 of the handbook.

This chapter provides information on the *peer probing* feature of NGCP that is available since mr5.4.1 release.

### 5.10.1 Introduction to Peer Probing Feature

The Sipwise NGCP provides web admin panel and API capabilities to configure peering servers in order to terminate calls to non-local subscribers. Those peering servers may become *temporarily unavailable* due to overloading or networking issues. The NGCP will fail over to another peering server (matching the corresponding peering rules) after a timeout configured at system level (see `sems.sbc.outbound_timeout` configuration parameter; 6 sec by default), if no provisional response (a response with a code in the range of 100 to 199) is received for the outbound INVITE request.

Even if this timer is set much lower, like 3 sec, the call setup time is increased significantly. This is even more true if multiple peering servers fail at the same time, which will sum up the individual timeouts, finally *causing call setup times reach the order of tens of seconds*.

To optimize the call setup time in such scenarios, a new feature is implemented to *continuously probe peering servers* via SIP messages, and mark them as unavailable on timeout or when receiving unexpected response codes. Appropriate SIP response codes from the peering servers will mark them as available again.

Peering servers *marked as unavailable* are then *skipped during call routing* in the peering selection process, which significantly shortens the call setup times if peering servers fail.

## 5.10.2 Configuration of Peer Probing

The system administrator has to configure the peer probing feature in 2 steps:

1. System-level configuration enables the peer probing feature in general on the NGCP and determines the operational parameters, such as timeouts, the SIP method used for probing requests, etc.
2. Peering server configuration will add / remove a peering server to the list of probed endpoints.

### 5.10.2.1 System-level Configuration

The parameters of peer probing are found in the main system configuration file `/etc/ngcp-config/config.yml`. You can see the complete list of configuration parameters in Section B.1.16 of the handbook, while the most significant ones are discussed here.

**Enabling peer probing** system-wide happens through the `kamailio.proxy.peer_probe.enable` parameter. If it is set to *yes* (which is the default value) then NGCP will consider probing of individual peering servers based on their settings.

**Timeout** of a single probing request can be defined through `kamailio.proxy.peer_probe.timeout` parameter. This is a value interpreted as seconds while NGCP will wait for a SIP response from the peering server. Default is 5 seconds.

The **probing interval** can be set through the `kamailio.proxy.peer_probe.interval` parameter. This is the time period in seconds that determines how often a probing request is sent to the peering servers. Default is 10 seconds.

The **SIP method** used for probing requests can be defined through `kamailio.proxy.peer_probe.method` parameter. Allowed values are: `OPTIONS` (default) and `INFO`.

---

#### Tip

The system administrator, in most of the cases, will not need to modify the default configuration values other than that of timeout and interval.

---

**If no available peering server is found, the call is rejected** with the response code and reason configured in `kamailio.proxy.early_rejects.peering_unavailable.announce_code` and `kamailio.proxy.early_rejects.peering_unavailable.announce_reason`. If a sound file is configured within the *system sound set* assigned to the calling party, an announcement is played as early media before the rejection.

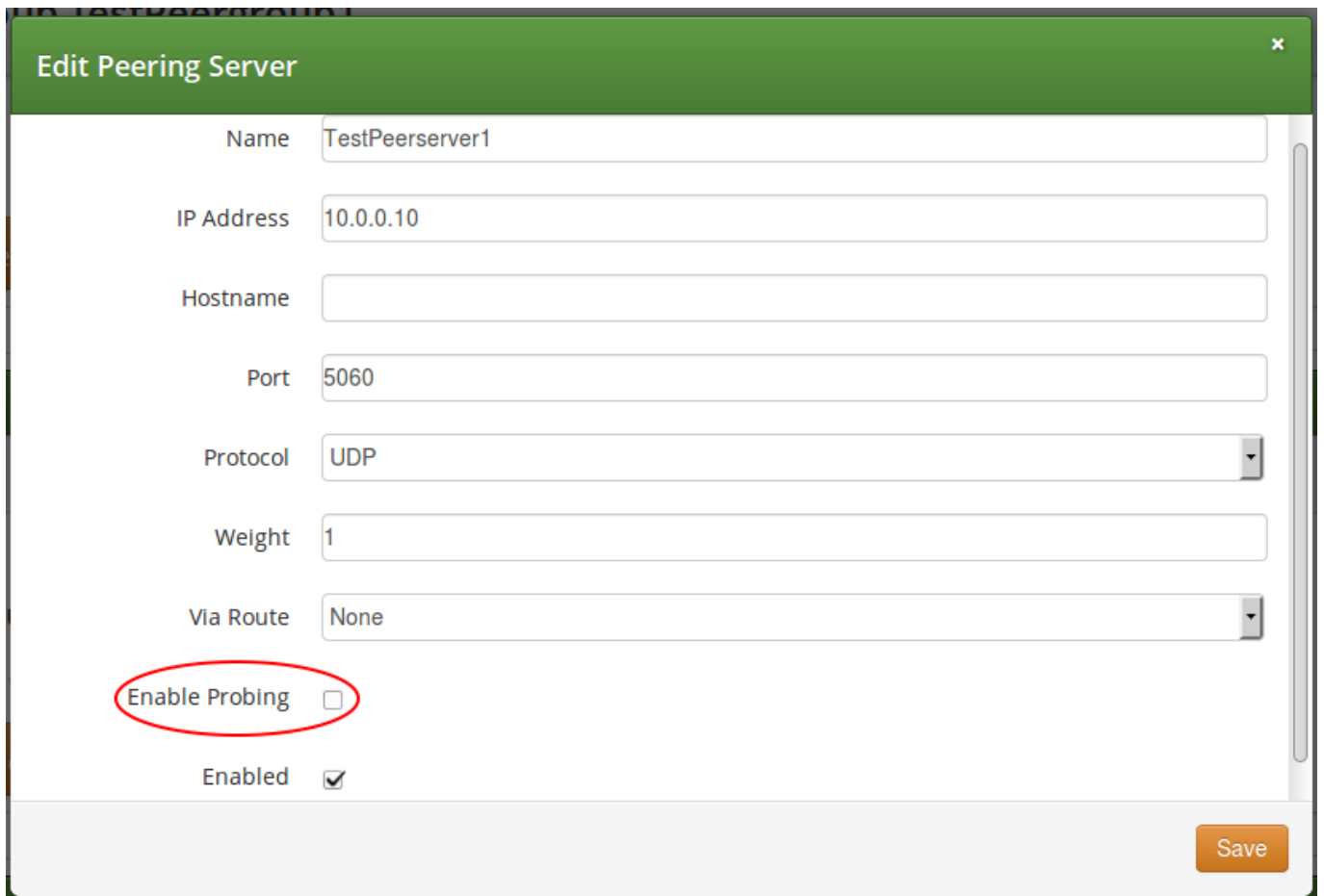
### 5.10.2.2 Individual Peering Server Configuration

When the peer probing feature is enabled on system-level, it is possible to add each individual peering server to the list of probed endpoints. You can change the probed status of a server in two ways:

#### ***Enable probing of a peering server via the admin web interface***

1. Open the properties panel of a peering server: *Peerings* → *select a peering group* → *Details* → *select a peering server* → *Edit*

2. Tick the checkbox *Enable Probing*
3. *Save changes*



**Edit Peering Server**

Name

IP Address

Hostname

Port

Protocol

Weight

Via Route

**Enable Probing**

Enabled

Save

Figure 50: Enable Probing of Peering Server

#### **Enable probing of a peering server via the REST API**

- when you *create a new peering server* you will use an HTTP *POST* request and the target URL:  
[https://<IP\\_of\\_NGCP>:1443/api/peeringservers](https://<IP_of_NGCP>:1443/api/peeringservers)
- when you *update an existing peering server* you will use an HTTP *PUT* or *PATCH* request and the target URL:  
[https://<IP\\_of\\_NGCP>:1443/api/peeringservers/id](https://<IP_of_NGCP>:1443/api/peeringservers/id)

In all cases you have to set the *probe* property to *true* in order to enable probing, and to *false* in order to disable probing. Default value is *false* and this property may be omitted in a create/update request, which ensures backward compatibility of the `/api/peeringservers` API resource.

### 5.10.3 Monitoring of Peer Probing

Peering server states, such as "reachable" / "unreachable", are continuously stored in a time-series database (InfluxDB type) by NGCP Proxy nodes. It is possible to **graphically represent the state of peering servers** on NGCP's admin web interface, just like other system variables (like CPU and memory usage, number of registered subscribers, etc.). However this is not available by default and must be configured by Sipwise.

State changes of peering servers are also reported by means of **SNMP traps**. Each time the reachable state of one of the monitored peering servers changes, NGCP will send an SNMP trap, raising or clearing the alarm.

The Sipwise MIB is extended by a table of peers per proxy, containing the peer ID and the peer name, along with the peer probe status. An external monitoring system can **poll the peers table via SNMP** to gather the peer status from each proxy's point of view.

The peer status can be obtained through the following route / OID:

```
...enterprises.sipwise.ngcp.ngcpObjects.ngcpMonitor.ngcpMonitorPeering.psTable.psEntry. ←  
    psPeerStatus  
  
.1.3.6.1.4.1.34274.1.1.2.40.2.1.7
```

Value of *psPeerStatus* can be:

- 0: unknown
- 1: administratively down
- 2: administratively up
- 3: probed, pending
- 4: probed, down
- 5: probed, up

### 5.10.4 Further Details for Advanced Users

---

#### Tip

This subchapter of the handbook is targeted on advanced system operators and Sipwise engineers and is not necessary to read in order to properly manage peer probing feature of NGCP.

---

#### 5.10.4.1 Behaviour of Kamailio Proxy Instances

Each *kamailio-proxy* instance on the proxy nodes performs the probing individually for performance reasons. Each proxy holds its result in its cache to avoid central storage and replication of the probing results. Each proxy will send an SNMP trap if it detects a state change for a peering server, because proxies might be geographically distributed along with their load-balancers and can therefore experience different probing results.

Each peering server is cross-checked against the hash table filled during outbound probing requests and is skipped by call routing logic, if a match is found.

On start or restart of the *kamailio-proxy* instance, the probing will start after the first interval, and NOT immediately after start. In the first probing interval the proxy will always try to send call traffic to peering servers until the first probing round is finished, and will only then start to skip unavailable peering servers.

#### 5.10.4.2 Changes to Kamailio Proxy Configuration

A new configuration template: `/etc/ngcp/config/templates/etc/kamailio/proxy/probe.cfg.tt2` is introduced to handle outbound probing requests.

#### 5.10.4.3 Database Changes

A new DB column: `provisioning.voip_peer_hosts.probe` with type `TINYINT(1)` (boolean) is added to the DB schema.

A peer status change will populate the `kamailio.dispatcher` table, inserting the SIP URI in format `sip:$sip:$port;transport=$transport` in dispatcher group 100, which defines the probing group for peering servers.

Also the `kamailio.dispatcher.attrs` column is populated with a parameter `peerid=$id`. This ID is used during probing to load the peer preferences: `outbound_socket` and `lbrtp_set`, that are required to properly route the probing request.

## 5.11 Fax Server

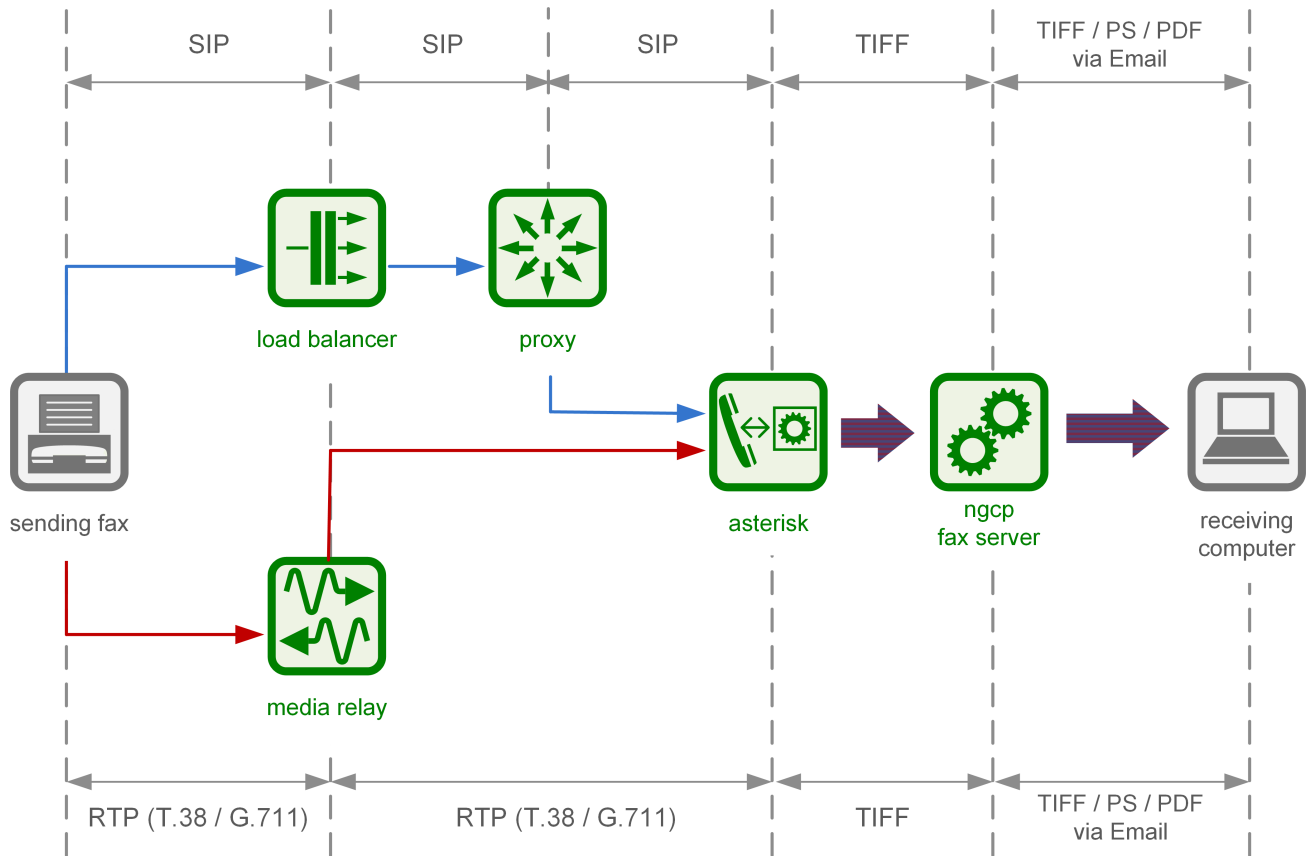
There is a Fax Server included in the sip:carrier. The following sections describe its architecture.

The Fax Server is included on the platform and requires no additional hardware. It supports both T38 and G711 codecs and provides a cost-effective paper-free office solution.

For the details of Fax Server configuration options, please see [Faxserver Configuration](#) Appendix C chapter in this handbook.

### 5.11.1 Fax2Mail Architecture

To receive faxes via email, a phone call from a sender is connected to the fax application module (Asterisk + NGCP Fax Server) on the sip:carrier. The received fax document is converted to the format the receiver has configured (either PS, PDF or TIFF) via the components outlined in the figure below. The email is delivered to one or more configured addresses.

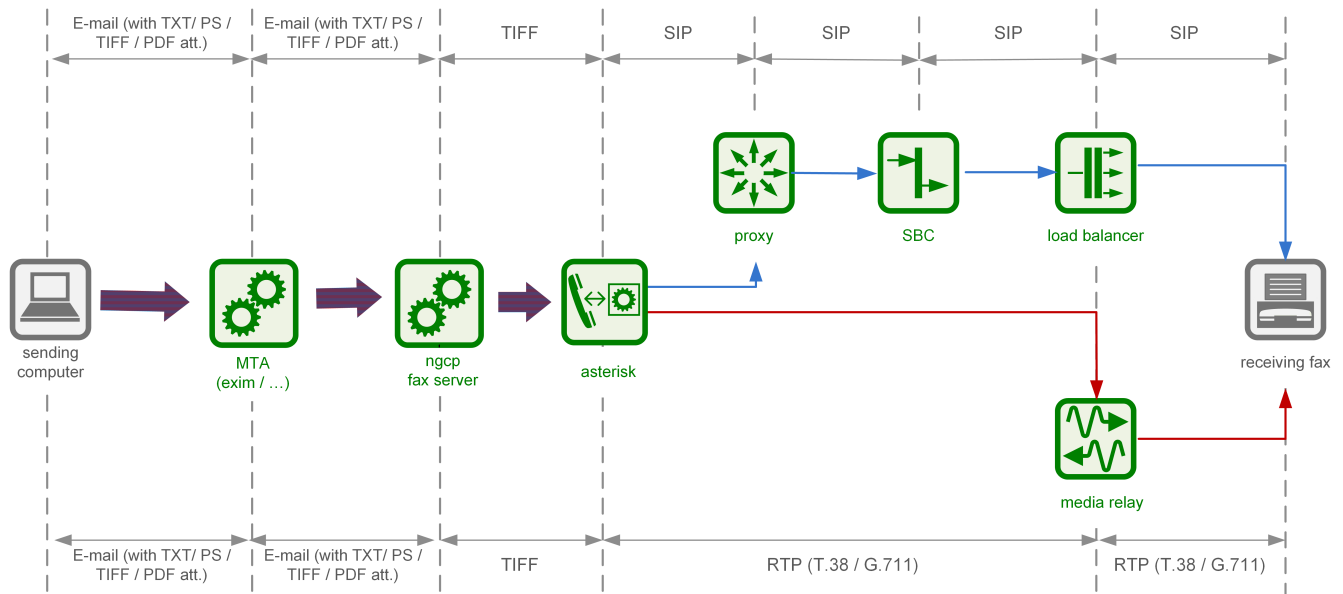


### 5.11.2 Sendfax and Mail2Fax Architecture

To send faxes via the sip:carrier a sender can use any email client or an interface such as Webfax or REST API.

Currently, supported formats are TXT, PS, TIFF and PDF.

The document is sent to the NGCP Fax Server instance on the sip:carrier. Once successfully queued by the fax server, it is converted to an internal TIFF format and is sent via the components outlined in the below figure to the specified phone number. Of course, a fax device that can receive the document must be connected on the destination side.



## 5.12 Voicemail System

### 5.12.1 Accessing the IVR Menu

For a subscriber to manage his voicebox via IVR, there are two ways to access the voicebox. One is to call the URI `voicebox@yourdomain` from the subscriber itself, allowing password-less access to the IVR, as the authentication is already done on SIP level. The second is to call the URI `voiceboxpass@yourdomain` from any number, causing the system to prompt for a mailbox and the PIN. The PIN can be set in the *Voicemail and Voicebox* section of the *Subscriber Preferences*.

#### 5.12.1.1 Mapping numbers and codes to IVR access

Since access might need to be provided from external networks like PSTN/Mobile, and since certain SIP phones do not support calling alphanumeric numbers to dial `voicebox`, you can map any number to the voicebox URIs using rewrite rules.

To do so, you can provision a match pattern e.g. `^(00|\+ )12345$` with a replace pattern `voicebox` or `voiceboxpass` to map a number to either password-less or password-based IVR access respectively. Create a new rewrite rule with the `Inbound` direction and the `Callee` field in the corresponding rewrite rule set.

For inbound calls from external networks, assign this rewrite rule set to the corresponding incoming peer. If you also need to map numbers for on-net calls, assign the rewrite rule set to subscribers or the whole SIP domain.

#### 5.12.1.2 External IVR access

When reaching `voiceboxpass`, the subscriber is prompted for her mailbox number and a password. All numbers assigned to a subscriber are valid input (primary number and any alias number). By default, the required format is in E.164, so the subscriber needs to enter the full number including country code, for example `4912345` if she got assigned a German number.

You can globally configure a rewrite rule in `config.yml` using `asterisk.voicemail.normalize_match` and `asterisk`

`isk.voicemail.normalize_replace`, allowing you to customize the format a subscriber can enter, e.g. having `^0 ([1-9] [0-9]+) $` as match part and `49$1` as replace part to accept German national format.

### 5.12.2 IVR Menu Structure

The following list shows you how the voicebox menu is structured.

- 1 Read voicemail messages
  - 3 Advanced options
    - \* 3 To Hear messages Envelope
    - \* \* Return to the main menu
  - 4 Play previous message
  - 5 Repeat current message
  - 6 Play next message
  - 7 Delete current message
  - 9 Save message in a folder
    - \* 0 Save in new Messages
    - \* 1 Save in old Messages
    - \* 2 Save in Work Messages
    - \* 3 Save in Family Messages
    - \* 4 Save in Friends Messages
    - \* # Return to the main menu
- 2 Change folders
  - 0 Switch to new Messages
  - 1 Switch to old Messages
  - 2 Switch to Work Messages
  - 3 Switch to Family Messages
  - 4 Switch to Friends Messages
  - # Get Back
- 3 Advanced Options
  - \* To return to the main menu
- 0 Mailbox options
  - 1 Record your unavailable message
    - \* 1 accept it
    - \* 2 Listen to it



- \* 3 Rerecord it
- 2 Record your busy message
  - \* 1 accept it
  - \* 2 Listen to it
  - \* 3 Rerecord it
- 3 Record your name
  - \* 1 accept it
  - \* 2 Listen to it
  - \* 3 Rerecord it
- 4 Record your temporary greetings
  - \* 1 accept it / or re-record if one already exist
  - \* 2 Listen to it / or delete if one already exist
  - \* 3 Rerecord it
- 5 Change your password
- \* To return to the main menu
- \* Help
- # Exit

### 5.12.3 Type Of Messages

A message/greeting is a short message that plays before the caller is allowed to record a message. The message is intended to let the caller know that you are not able to answer their call. It can also be used to convey other information like when you will be available, other methods to contact you, or other options that the caller can use to receive assistance.

The IVR menu has three types of greetings.

#### 5.12.3.1 Unavailable Message

The standard voice mail greeting is the "unavailable" greeting. This is used if you don't answer the phone and so the call is directed to your voice mailbox.

- You can record a custom unavailable greeting.
- If you have not recorded your unavailable greeting but have recorded your name, the system will play a generic message like: "Recorded name is unavailable."
- If you have not recorded your unavailable greeting, the phone system will play a generic message like: "Digits-of-number-dialed is unavailable".

### 5.12.3.2 Busy Message

If you wish, you can record a custom greeting used when someone calls you and you are currently on the phone. This is called your "Busy" greeting.

- You can record a custom busy greeting.
- If you have not recorded your busy greeting but have recorded your name, the phone system will play a generic message: "Recorded name is busy."
- If you have not recorded your busy greeting and have not recorded your name (see below), the phone system will play a generic message: "Digits-of-number-dialed is busy."

### 5.12.3.3 Temporary Greeting

You can also record a temporary greeting. If it exists, a temporary greeting will always be played instead of your "busy" or "unavailable" greetings. This could be used, for example, if you are going on vacation or will be out of the office for a while and want to inform people not to expect a return call anytime soon. Using a temporary greeting avoids having to change your normal unavailable greeting when you leave and when you come back.

## 5.12.4 Folders

The Voicemail system allows you to save and organize your messages into folders. There can be up to ten folders.

### 5.12.4.1 The Default Folder List

- 0 - New Messages
- 1 - Old Messages
- 2 - Work Messages
- 3 - Family Messages
- 4 - Friends Messages

When a caller leaves a message for you, the system will put the message into the "New Messages" folder. If you listen to the message, but do not delete the message or save the message to a different folder, it will automatically move the message to the "Old Messages" folder. When you first log into your mailbox, the Voicemail System will make the "New Messages" folder the current folder if you have any new messages. If you do not have any new messages it will make the "Old Messages" folder the current folder.

## 5.12.5 Voicemail Languages Configuration

To add a new language or to change the pronunciation for an existing one, ensure that **mode=new** is defined in `/etc/ngcp-config/templates/etc/asterisk/say.conf.tt2`. Adjust the configuration in the same file using the manual in the beginning. Then, as usual, make the new configuration active.

### 5.12.6 Flowcharts with Voice Prompts

This section shows flowcharts of calls to the voicemail system. Flowcharts contain the name of prompts as they are identified among *Asterisk* voice prompts.

5.12.6.1 Listening to New Messages

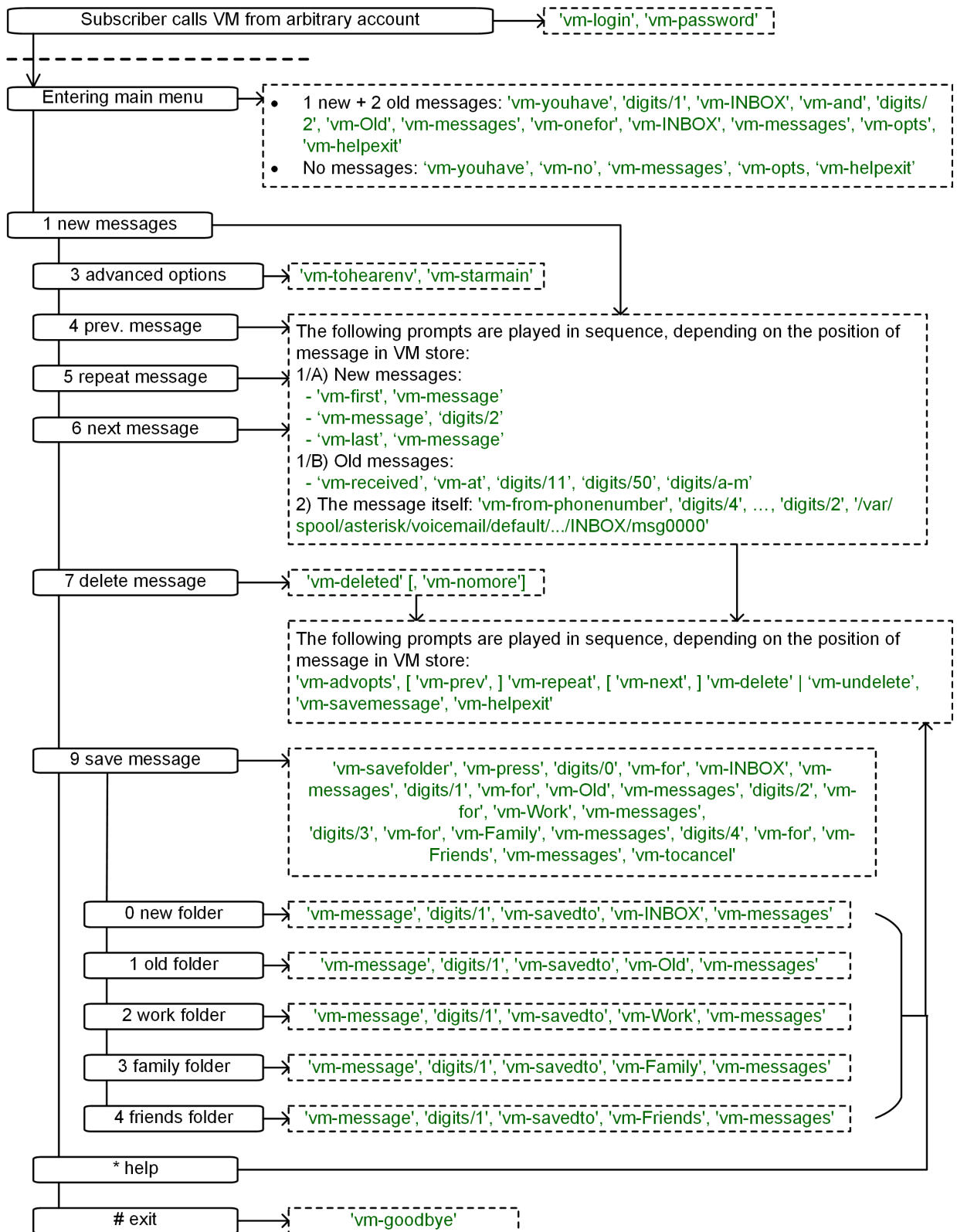


Figure 51: Flowchart of Listening to New Messages

### 5.12.6.2 Changing Voicemail Folders

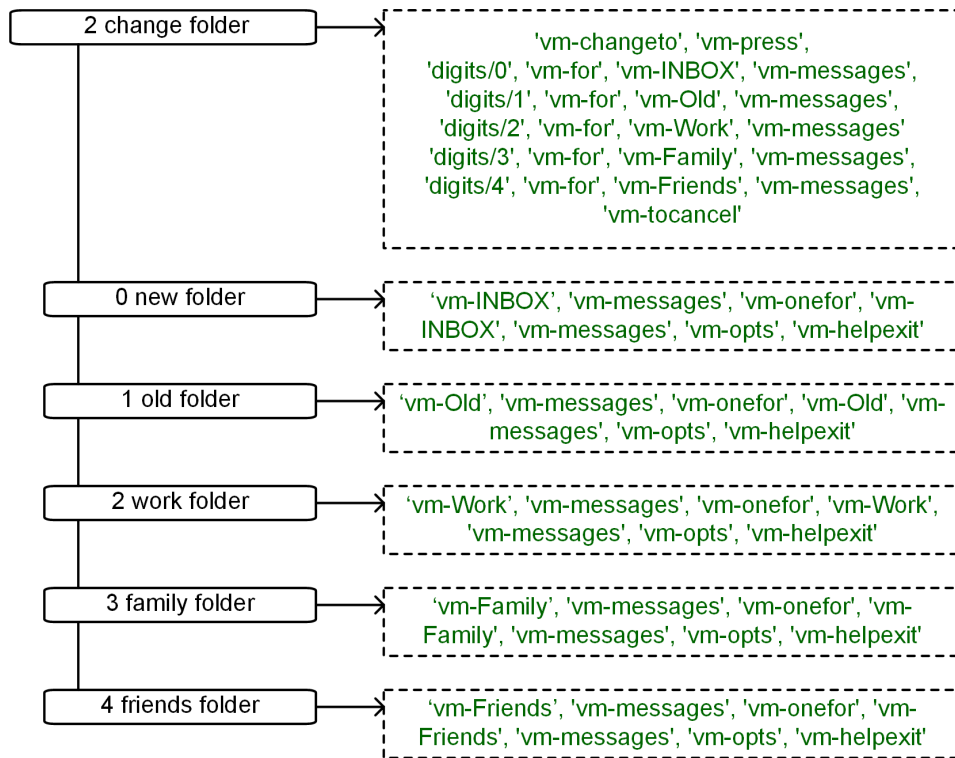


Figure 52: Flowchart of Changing Voicemail Folders

5.12.6.3 Mailbox Options

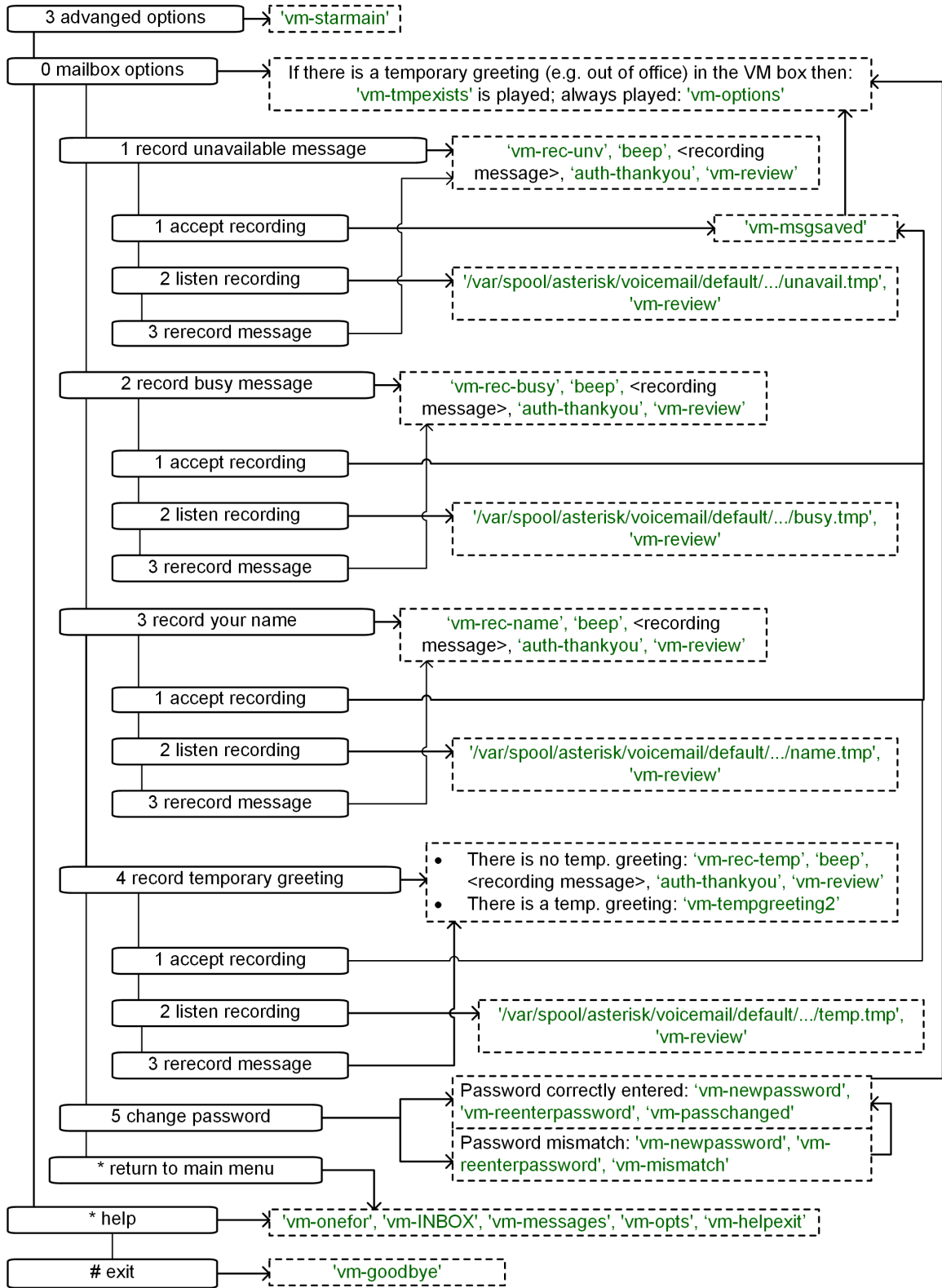


Figure 53: Flowchart of Changing Mailbox Options

5.12.6.4 Leaving a Message

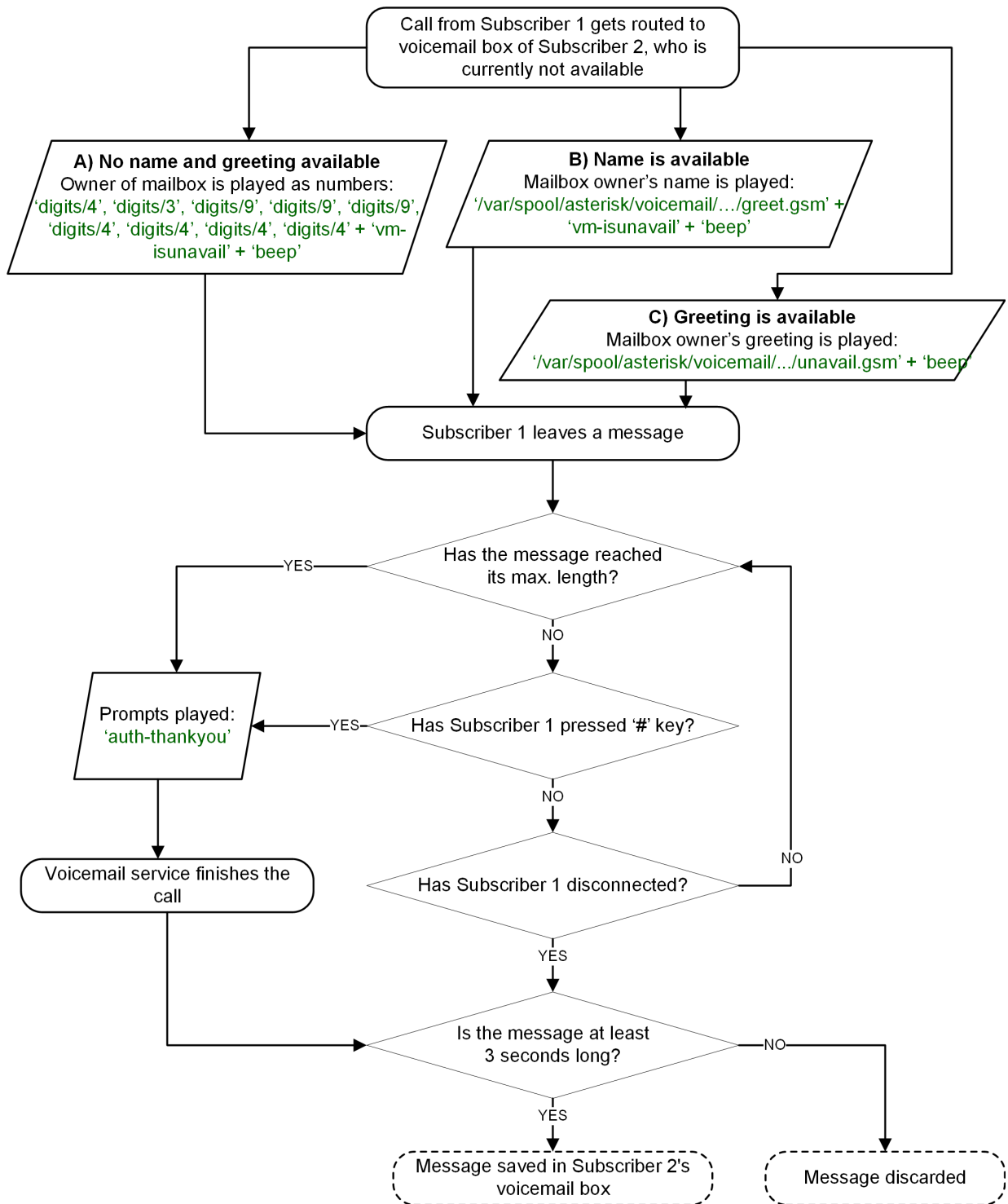
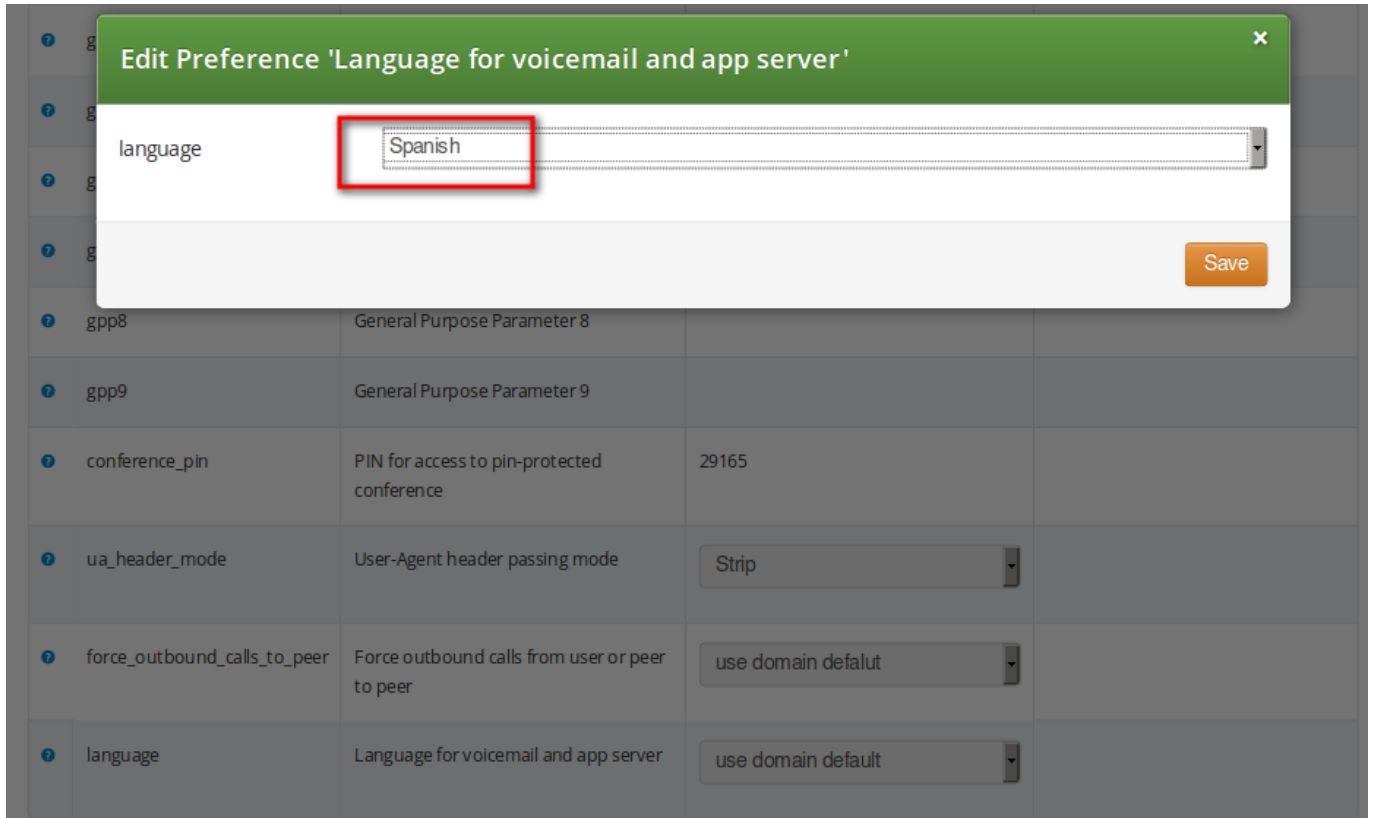


Figure 54: Flowchart of Leaving a Voice Message

### 5.13 Configuring Subscriber IVR Language

The language for the Voicemail system IVR or Vertical Service Codes (VSC) IVRs may be set using the subscriber or domain preference *language*.



The sip:carrier provides the pre-installed prompts for the Voicemail in the English, Spanish, French and Italian languages and the pre-installed prompts for the Vertical Service Codes IVRs in English only.


The other IVRs such as the Conference system and the error announcements use the Sound Sets configured in NGCP Panel and uploaded by the administrator in his language of choice.

### 5.14 Sound Sets

The sip:carrier provides the administrator with ability to upload the voice prompts such as conference prompts or call error announcements on the *Sound Sets* page. There is a preference *sound\_set* in the *NAT and Media Flow Control* section on Domain and Subscriber levels to link subscribers to the sound set that they should hear (as usual the subscriber preference overrides the domain one). Sound Sets can be defined in *Settings*→*Sound Sets*. To create a new Sound Set, click *Create Sound Set*. Then click the *Files* button.



Logged in as administrator | Language | Logout



Monitoring & Statistics
Settings

---

## Sound Sets

← Back
★ Create Sound Set

Sound set successfully created

Show  entries Search:

#	Reseller	Customer	Name	Description	
1	default		Conference		<span>Edit</span> <span>Delete</span> <span style="border: 2px solid red; padding: 2px;">Files</span>
2	default		Early media rejects	Failed call attempt announcements	

Showing 1 to 2 of 2 entries

← 1 →

**Note**

You may use 8 or 16 bit mono WAV audio files for all of the voice prompts.

**5.14.1 Configuring Early Reject Sound Sets**

The call error announcements are grouped under *Early Rejects* section. Unfold the section and click *Upload* next to the sound handles (Names) that you want to use. Choose a WAV file from your file system, and click the Loopplay setting if you want to play the file in a loop instead of just once. Click Save to upload the file.

early_rejects			
Name	Filename	Loop	
block_in		■	<input type="button" value="Upload"/>
block_out		■	
block_ncos		■	
block_override_pin_wrong		■	
locked_in		■	
locked_out		■	
max_calls_in		■	
max_calls_out		■	
max_calls_peer		■	
unauth_caller_ip		■	

The call error announcements are played to the user in early media hence the name "Early Reject". If you don't provide the sound files for any handles they will not be used and the sip:carrier will fallback to sending the error response code back to the user.

The exact error status code and text are configurable in the `/etc/ngcp-config/config.yml` file, in `kamailio.proxy.early_rejects` section. Please look for the announcement handle listed in below table in order to find it in the configuration file.

Table 3: Early Reject Announcements

Handle	Description	Message played
announce_before_cf	This is an announcement that the calling party hears before the call is being forwarded (Unconditional and Not Available cases) to the destination. The feature can be activated with Applications / <code>play_announce_before_cf</code> domain or subscriber preference.	N/A (custom message, no default)
block_in	This is what the calling party hears when a call is made from a number that is blocked by the incoming block list ( <code>adm_block_in_list</code> , <code>block_in_list</code> customer/subscriber preferences)	Your call is blocked by the number you are trying to reach.

Table 3: (continued)

Handle	Description	Message played
block_out	This is what the calling party hears when a call is made to a number that is blocked by the outgoing block list ( <i>adm_block_out_list</i> , <i>block_out_list</i> customer/subscriber preferences)	Your call to the number you are trying to reach is blocked.
block_ncos	This is what the calling party hears when a call is made to a number that is blocked by the NCOS level assigned to the subscriber or domain (the NCOS level chosen in <i>ncos</i> and <i>adm_ncos</i> preferences). <i>PLEASE NOTE:</i> It is not possible to configure the status code and text.	Your call to the number you are trying to reach is not permitted.
block_override_pin_wrong	Announcement played to calling party if it used wrong PIN code to override the outgoing user block list or the NCOS level for this call (the PIN set by <i>block_out_override_pin</i> and <i>adm_block_out_override_pin</i> preferences)	The PIN code you have entered is not correct.
callee_busy	Announcement played on incoming call to the subscriber which is currently busy (486 response from the UAS)	The number you are trying to reach is currently busy. Please try again later.
callee_offline	Announcement played on incoming call to the subscriber which is currently not registered	The number you are trying to reach is currently not available. Please try again later.
callee_tmp_unavailable	Announcement played on incoming call to the subscriber which is currently unavailable (408, other 4xx or no response code or 30x with malformed contact)	The number you are trying to reach is currently not available. Please try again later.
callee_unknown	Announcement that is played on call to unknown or invalid number (not associated with any of our subscribers/hunt groups)	The number you are trying to reach is not in use.
cf_loop	Announcement played when the called subscriber has the call forwarding configured to itself	The number you are trying to reach is forwarded to an invalid destination.

Table 3: (continued)

Handle	Description	Message played
emergency_geo_unavailable	Announcement played when emergency destination is dialed but the destination is not provisioned for the location of the user. <i>PLEASE NOTE:</i> The configuration entry for this case in <code>/etc/ngcp-config/config.yml</code> file is <code>emergency_invalid</code> .	The emergency number you have dialed is not available in your region.
emergency_unsupported	Announcement played when emergency destination is dialed but the emergency calls are administratively prohibited for this user or domain ( <i>reject_emergency</i> preference is enabled)	You are not allowed to place emergency calls from this line. Please use a different phone.
error_please_try_later	Announcement played when the call is handled by 3rd party call control (PCC) and there was an error during call processing. <i>PLEASE NOTE:</i> This announcement may be configured in the sound set in <code>voucher_recharge</code> section.	An error has occurred. Please try again later.
invalid_speeddial	This is what the calling party hears when it calls an empty speed-dial slot	The speed dial slot you are trying to use is not available.
locked_in	Announcement played on incoming call to a subscriber that is locked for incoming calls	The number you are trying to reach is currently not permitted to receive calls.
locked_out	Announcement played on outgoing call to subscriber that is locked for outgoing calls	You are currently not allowed to place outbound calls.
max_calls_in	Announcement played on incoming call to a subscriber who has exceeded the <i>concurrent_max</i> limit by sum of incoming and outgoing calls or whose customer has exceeded the <i>concurrent_max_per_account</i> limit by sum of incoming and outgoing calls	The number you are trying to reach is currently busy. Please try again later.
max_calls_out	Announcement played on outgoing call to a subscriber who has exceeded the <i>concurrent_max</i> (total limit) or <i>concurrent_max_out</i> (limit on number of outbound calls) or whose customer has exceeded the <i>concurrent_max_per_account</i> or <i>concurrent_max_out_per_account</i> limit	All outgoing lines are currently in use. Please try again later.

Table 3: (continued)

Handle	Description	Message played
max_calls_peer	Announcement played on calls from the peering if that peer has reached the maximum number of concurrent calls (configured by admin in <i>concurrent_max</i> preference of peering server). <i>PLEASE NOTE</i> : There is no configuration option of the status code and text in <i>config.yml</i> file for this case.	The network you are trying to reach is currently busy. Please try again later.
no_credit	Announcement played when prepaid account has insufficient balance to make a call to this destination	You don't have sufficient credit balance for the number you are trying to reach.
peering_unavailable	Announcement played in case of outgoing off-net call when there is no peering rule matching this destination and/or source	The network you are trying to reach is not available.
reject_vsc	When the VSC (Vertical Service Code) service is disabled in domain or subscriber preferences (Access Restrictions / <i>reject_vsc</i> is set to TRUE) and a subscriber tries to make a call with VSC, an announcement is played.	N/A (custom message, no default)
relaying_denied	Announcement played on inbound call from trusted IP (e.g. external PBX) with non-local Request-URI domain	The network you are trying to reach is not available.
unauth_caller_ip	This is what the calling party hears when it tries to make a call from unauthorized IP address or network ( <i>allowed_ips</i> , <i>man_allowed_ips</i> preferences)	You are not allowed to place calls from your current network location.
voicebox_unavailable	<i>PLEASE NOTE</i> : This announcement is already obsolete, as of NGCP version mr5.3	The voicemail of the number you are trying to reach is currently not available. Please try again later.

There are some early reject scenarios when either **no voice announcement is played, or a fixed announcement is played**. In either case a SIP error status message is sent from NGCP to the calling party. It is possible to configure the exact status code and text for such cases in the */etc/ngcp-config/config.yml* file, in *kamailio.proxy.early\_rejects* section. The below table gives an overview of those early reject cases.

Table 4: Additional Early Reject Reason Codes

Handle	Description
block_admin	Caller blocked by adm_block_in_list, adm_block_in_clir and callee blocked by adm_block_out_list (customer or subscriber preference)
block_callee	Callee blocked by subscriber preference block_out_list
block_caller	Caller blocked by subscriber preference block_in_list, block_in_clir
block_contract	Caller blocked by customer preference block_in_list, block_in_clir and callee blocked by customer preference block_out_list
callee_tmp_unavailable_gp	Callee is a PBX group with 0 members. Announcement callee_tmp_unavailable is played; status code and text can be configured.
callee_tmp_unavailable_tm	Callee is a PBX group and we have a timeout (i.e. no group member could be reached). Announcement callee_tmp_unavailable is played; status code and text can be configured.
emergency_invalid	<i>PLEASE NOTE:</i> This handle refers to the same early reject case as emergency_geo_unavailable, but is labeled differently in the configuration file.

## 5.15 Conference System

The sip:carrier provides the simple pin-protected conferencing service built using the SEMS DSM scripting language. Hence it is open for all kinds of modifications and extensions.

Template files for the sems conference scripts stored in `/etc/ngcp-config/templates/etc/ngcp-sems/`:

- IVR script: `/etc/ngcp-config/templates/etc/ngcp-sems/dsm/confpin.dsm.tt2`
- Config: `/etc/ngcp-config/templates/etc/ngcp-sems/dsm/confpin.conf.tt2`

### 5.15.1 Configuring Call Forward to Conference

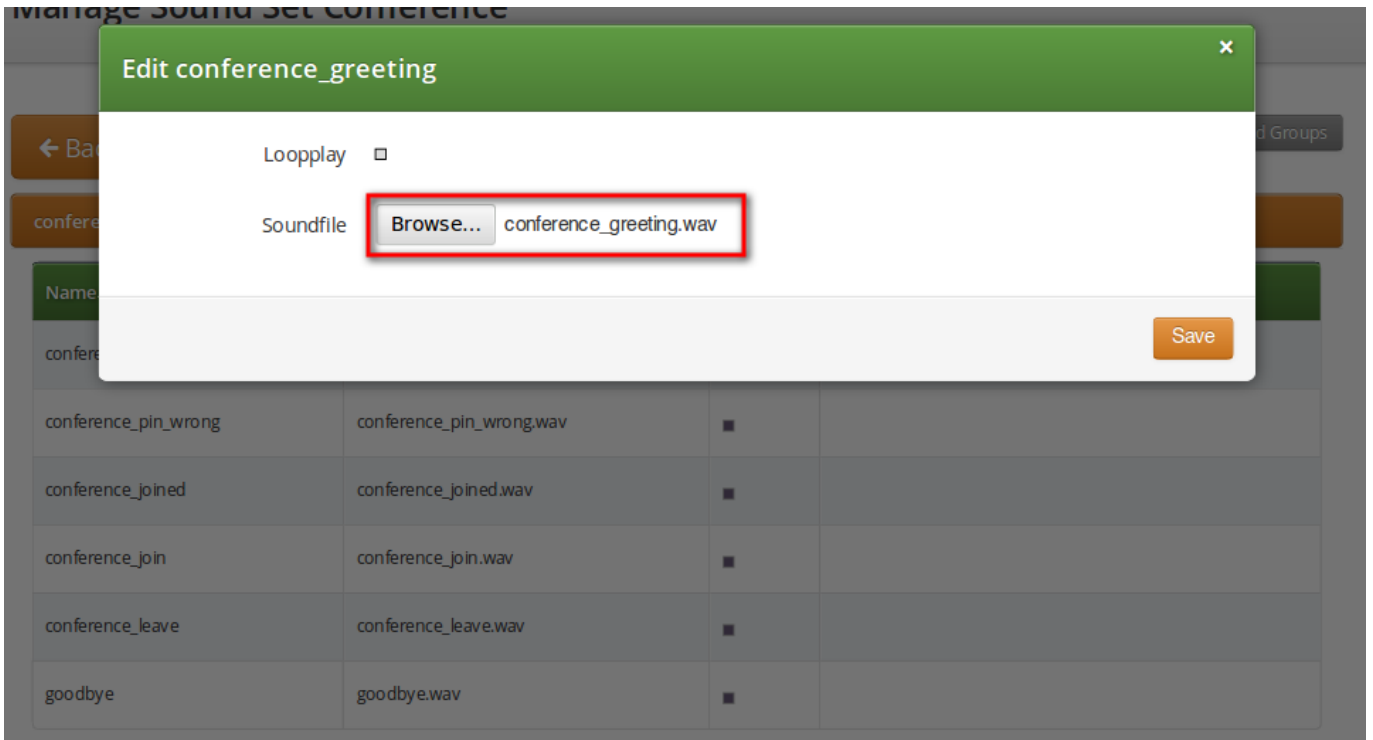
Go to your *Subscriber Preferences* and click *Edit* on the Call Forward Type you want to set (e.g. *Call Forward Unconditional*).

The screenshot shows a modal dialog titled "Edit Call Forward Unconditional". The "Destination" field has three radio button options: "Voicemail", "Conference" (which is selected and highlighted with a red rectangular box), and "URI/Number". Below this, the "URI/Number" field is an empty text input. The "for (seconds)" field contains the value "300". At the bottom right of the dialog are two buttons: "Advanced View" and "Save". The background is a blurred view of a web interface showing a table of call forward types, including "Call Forward Busy", "Call Forward Timeout", and "Call Forward Unavailable".

You should select *Conference* option in the *Destination* field and leave the *URI/Number* empty. The timeout defines for how long this destination should be tried to ring.

### 5.15.2 Configuring Conference Sound Sets

Sound Sets can be defined in *Settings*→*Sound Sets*. To create a new Sound Set, click *Create Sound Set*. Then click the *Files* button.



Upload the following files:

Table 5: Conference Sound Sets

Handle	Message played
conference_greeting	Welcome to the conferencing service.
conference_pin	Please enter your PIN, followed by the pound key.
conference_pin_wrong	You have entered an invalid PIN number. Please try again.
conference_joined	You will be placed into the conference.
conference_first	You are the first person in the conference.
conference_join	A person has joined the conference.
conference_leave	A person has left the conference.
conference_max_participants	All conference lines are currently in use. Please try again later.
conference_waiting_music	... waiting music...
goodbye	Goodbye.

**Note**

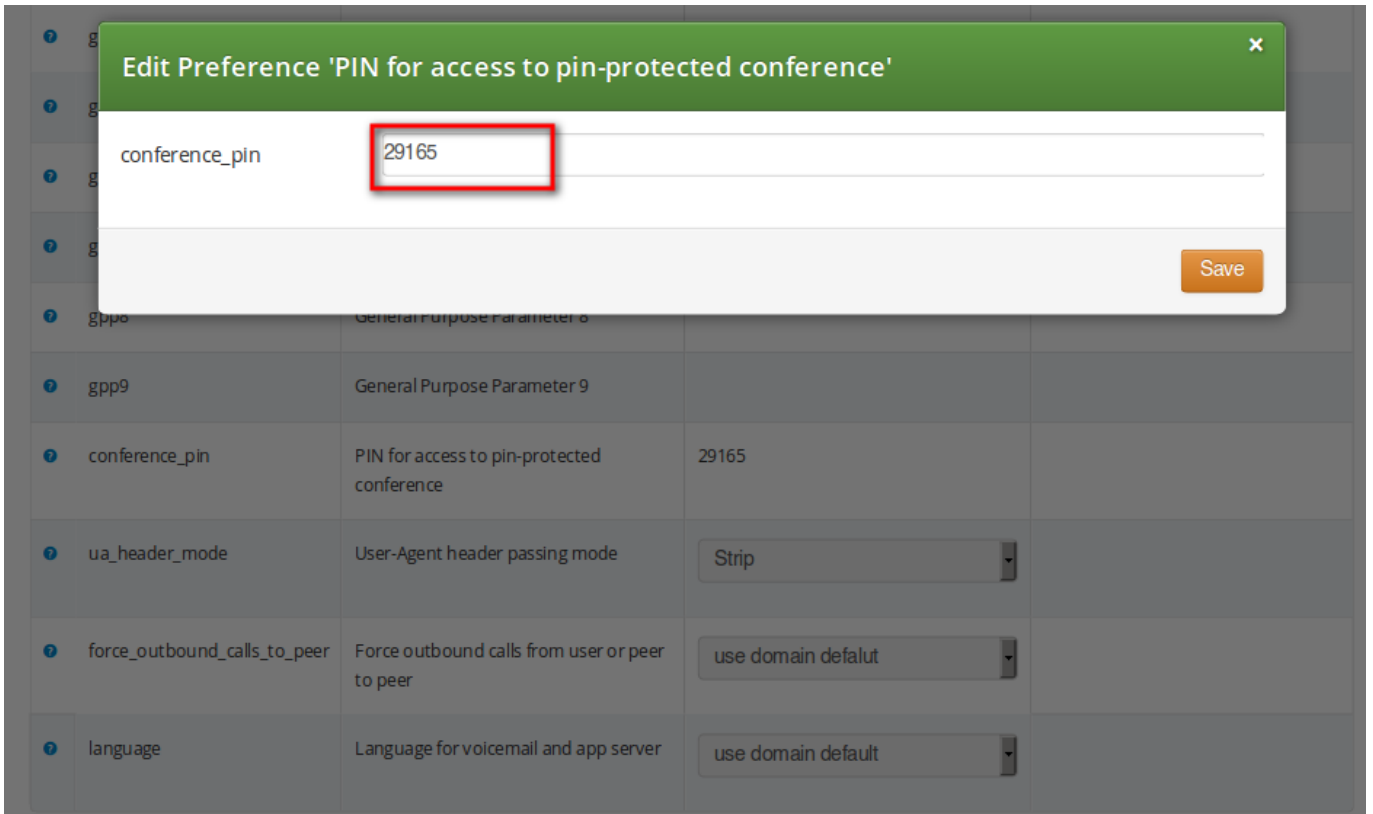
You may use 8 or 16 bit mono WAV audio files.

Then set the preference *sound\_set* on the Domain or Subscriber level in order to assign the Sound Set you have just created to the subscriber (as usual the subscriber preference overrides the domain one).



### 5.15.3 Joining the Conference

There are 2 ways of joining a conference: with or without PIN code. The actual way of joining the conference depends on *Subscriber* settings. A subscriber who has activated the conference through call forwarding may set a PIN in order to protect the conference from unauthorized access. To activate the PIN one has to enter a value in *Subscriber* → *Details* → *Preferences* → *Internals* → *conference\_pin* field.



The image shows a web interface for editing preferences. A modal dialog titled "Edit Preference 'PIN for access to pin-protected conference'" is open. Inside the dialog, there is a text input field labeled "conference\_pin" containing the value "29165". A red rectangular box highlights the input field. To the right of the input field is an orange "Save" button. Below the dialog, a table of preferences is visible, with the "conference\_pin" row highlighted. The table has columns for the preference name, description, and value.

Preference Name	Description	Value
gpp0	General Purpose Parameter 0	
gpp9	General Purpose Parameter 9	
conference_pin	PIN for access to pin-protected conference	29165
ua_header_mode	User-Agent header passing mode	Strip
force_outbound_calls_to_peer	Force outbound calls from user or peer to peer	use domain default
language	Language for voicemail and app server	use domain default

Figure 55: Setting Conference PIN

In case the PIN protection for the conference is activated, when someone calls the subscriber who has enabled the conference, the caller is prompted to enter the PIN of the conference. Upon the successful entry of the PIN the caller hears the announcement that he is going to be placed into the conference and at the same time this is announced to all participants already in the conference.

### 5.15.4 Conference Flowchart with Voice Prompts

The following 2 sections show flowcharts with voice prompts that are played to a caller when he dials the conference.

5.15.4.1 Conference Flowchart with PIN Validation

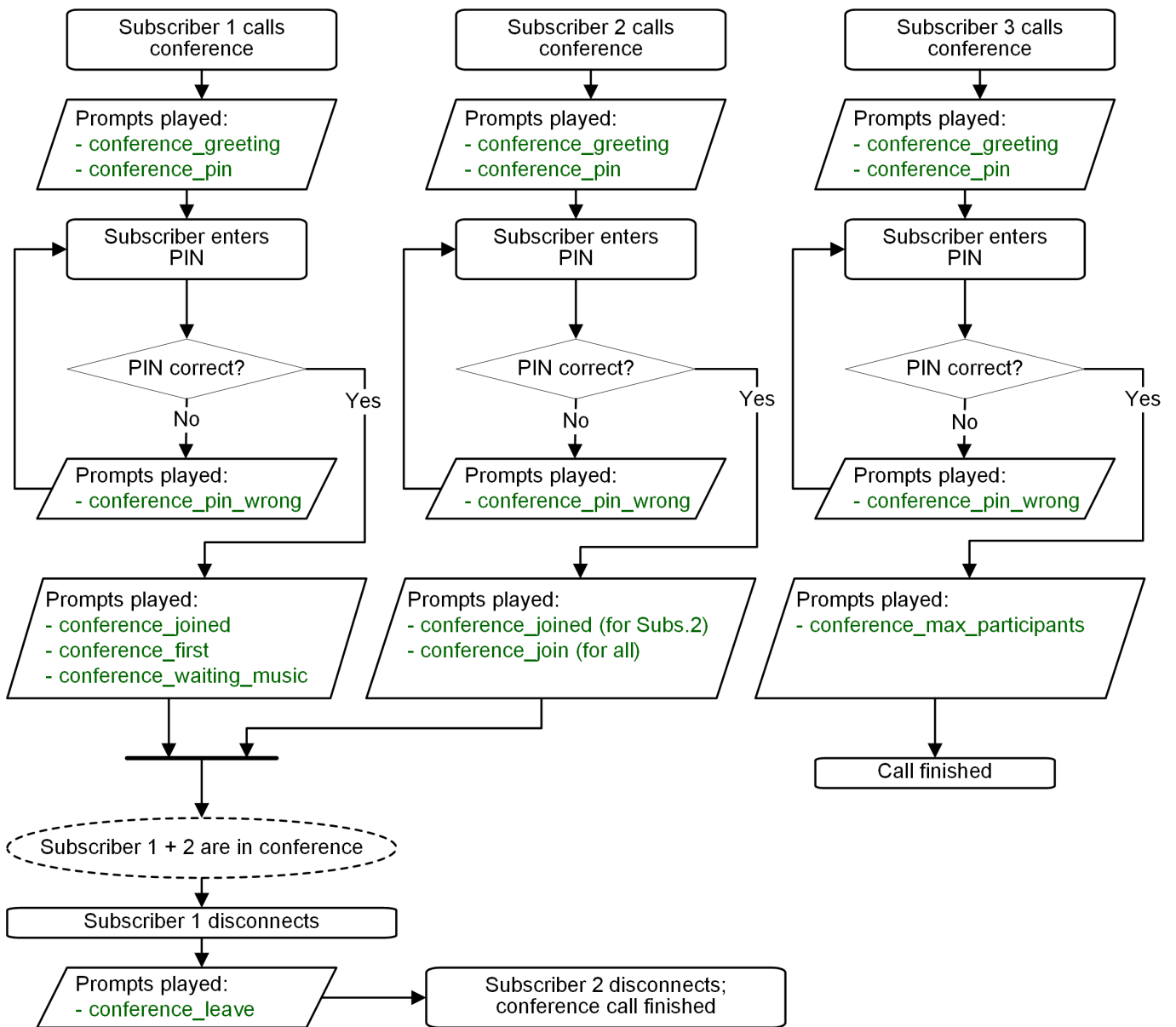


Figure 56: Flowchart of Conference with PIN Validation

### 5.15.4.2 Conference Flowchart without PIN

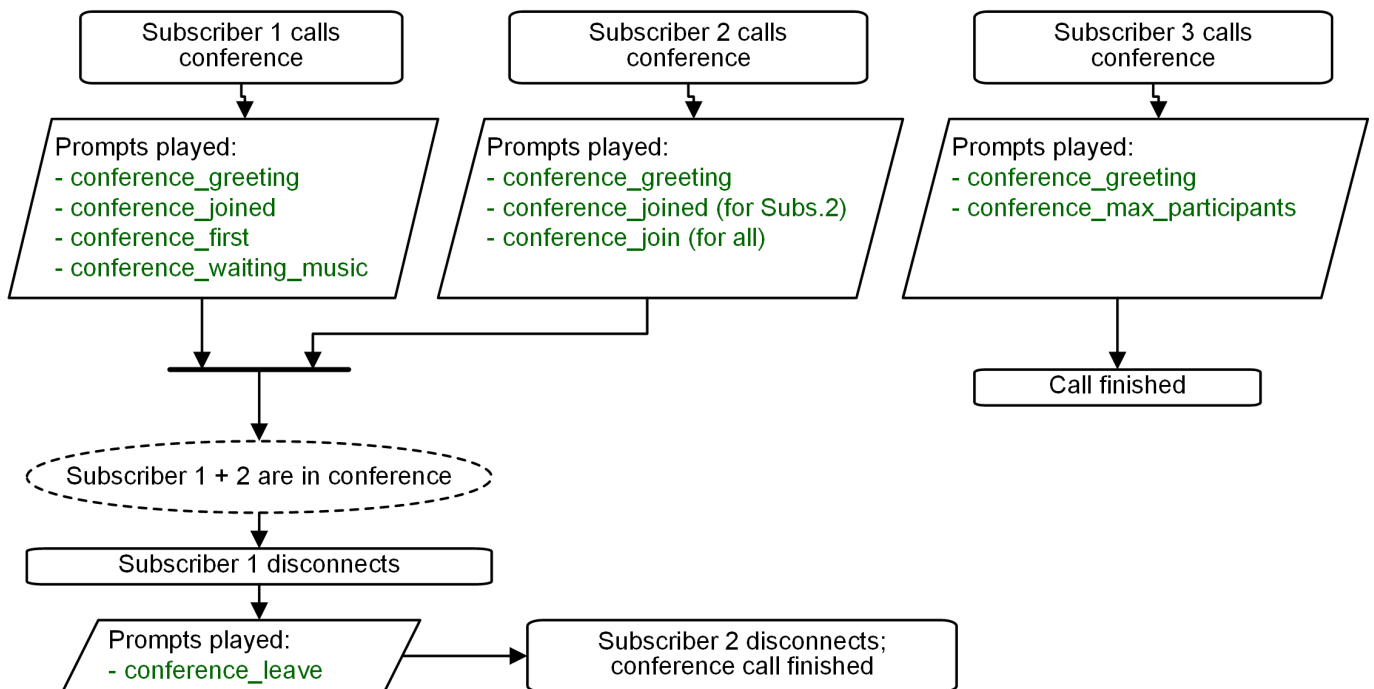


Figure 57: Flowchart of Conference without PIN

## 5.16 Malicious Call Identification (MCID)

MCID feature allows customers to report unwanted calls to the platform operator.

### 5.16.1 Setup

To enable the feature first edit `config.yml` and enable there `apps: malicious_call: yes` and `kamailio: store_recentcalls: yes`. The latter option enables kamailio to store recent calls per subscriber UUID in the redis DB (the amount of stored recent calls will not exceed the amount of provisioned subscribers).

Next step is to create a system sound set for the feature. In *Settings* → *Sound Sets* either use your already existing *Sound Set* or create a new *Sound Set* and then assign it to your domain or subscribers. In the *Sound Set* there is a fileset *malicious\_call\_identification* → for that purpose.

Once the *Sound Set* is created the Subscriber's Preferences *Malicious Call Identification* must be enabled under *Subscriber* → *Preferences* → *Applications* menu. The same parameter can be set in the Customer's preferences to enable this feature for all its subscribers.

The final step is to create a new *Rewrite Rule* and to route calls to, for instance `*123` → MCID application. For that you create a *Callee Inbound* rewrite rule `^(\\*123)$ → malicious_call`

Finally you run `ngcpcfg apply "Enabling MCID"` to recreate the templates and automatically restart depended services.

### 5.16.2 Usage

As a subscriber, to report a malicious call you call to either *malicious\_call* or to your custom number assigned for that purpose. Please note that you can report only your last received call. You will hear the media reply from the *Sound Set* you have previously configured.

To check reported malicious calls as the platform operator open *Settings*→*Malicious Calls* tab where you will see a list of registered calls. You can selectively delete records from the list and alternatively you can manage the reported calls by using the REST API.

### 5.16.3 Advanced configuration

By default the expiration time for the most recent call per subscriber is 3600 seconds (1 hour). If you wish to prolong or shorten the expiration time open `constants.yml` and set there `recentcalls: expire: 3600` to a new value, and issue `ngcpcfg apply "Enabling MCID" afterwards`.

## 5.17 Subscriber Profiles

The preferences a subscriber can provision by himself via the CSC can be limited via profiles within profile sets assigned to subscribers.

### 5.17.1 Subscriber Profile Sets

Profile sets define containers for profiles. The idea is to define profile sets with different profiles by the administrator (or the reseller, if he is permitted to do so). Then, a subscriber with administrative privileges can re-assign profiles within his profile sets for the subscribers of his customer account.

Profile Sets can be defined in *Settings*→*Subscriber Profiles*. To create a new Profile Set, click *Create Subscriber Profile Set*.

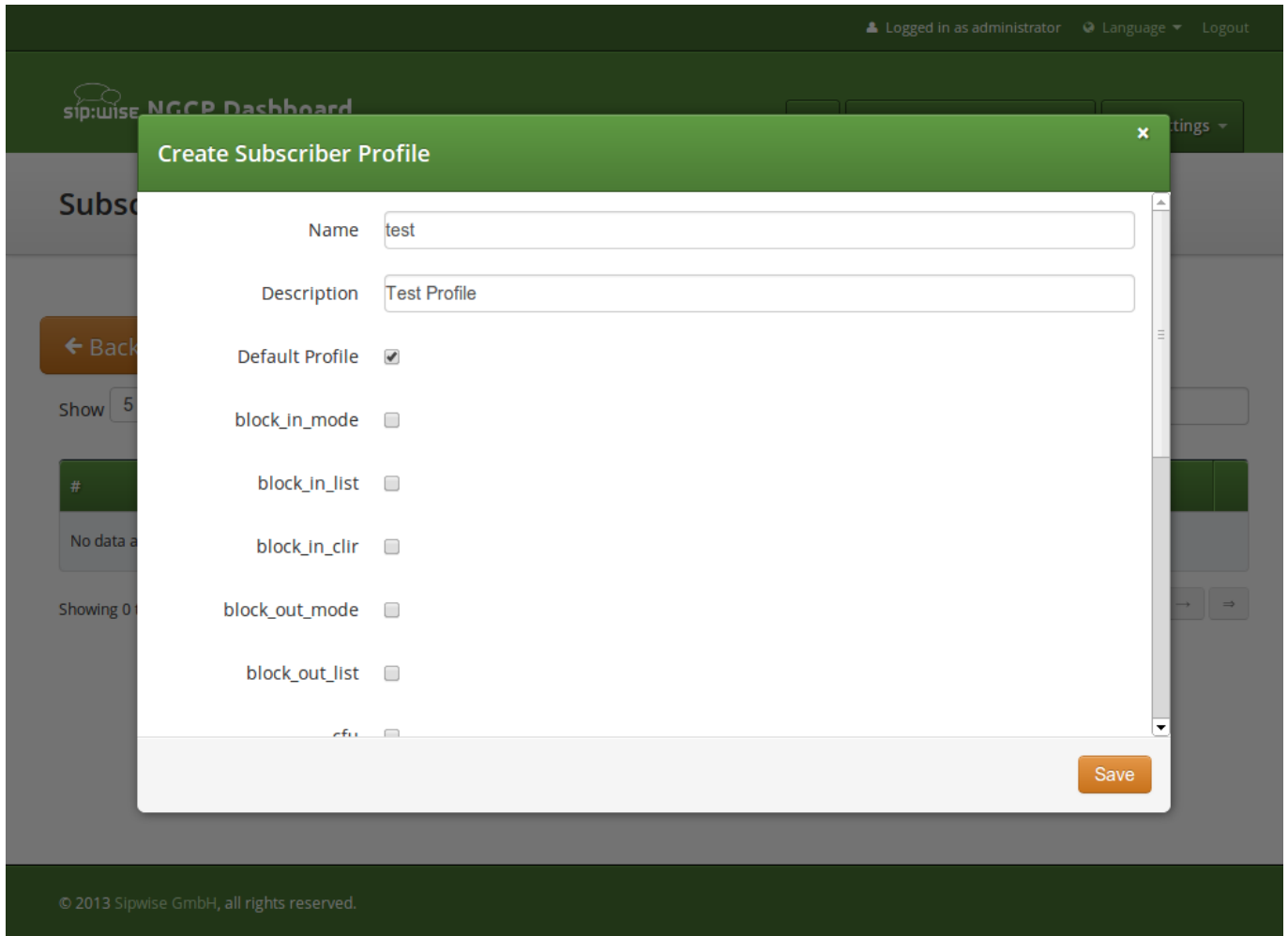
The screenshot displays the sip:wise NGCP Dashboard interface. At the top, it shows the user is logged in as an administrator, with options for language and logout. The dashboard title is 'sip:wise NGCP Dashboard'. Below the title, there are navigation buttons for 'Monitoring & Statistics' and 'Settings'. The main content area is partially obscured by a modal window titled 'Create Subscriber Profile Sets'. This modal window contains a 'Reseller' section with a search bar and a table of resellers. The table has columns for '#', 'Name', 'Contract #', 'Status', and a checkbox. Two resellers are listed: 'default' (Contract # 1, Status active) and 'test' (Contract # 2, Status active). Below the table, it says 'Showing 1 to 2 of 2 entries' and includes pagination controls. A 'Create Reseller' button is present. The modal also has input fields for 'Name' (containing 'testset') and 'Description' (containing 'Test Set'). A 'Save' button is located at the bottom right of the modal. The footer of the dashboard reads '© 2013 Sipwise GmbH, all rights reserved.'

#	Name	Contract #	Status	
1	default	1	active	<input checked="" type="checkbox"/>
2	test	2	active	<input type="checkbox"/>

You need to provide a reseller, name and description.

To create Profiles within a Profile Set, hover over the Profile Set and click the *Profiles* button.

Profiles within a Profile Set can be created by clicking the *Create Subscriber Profile* button.



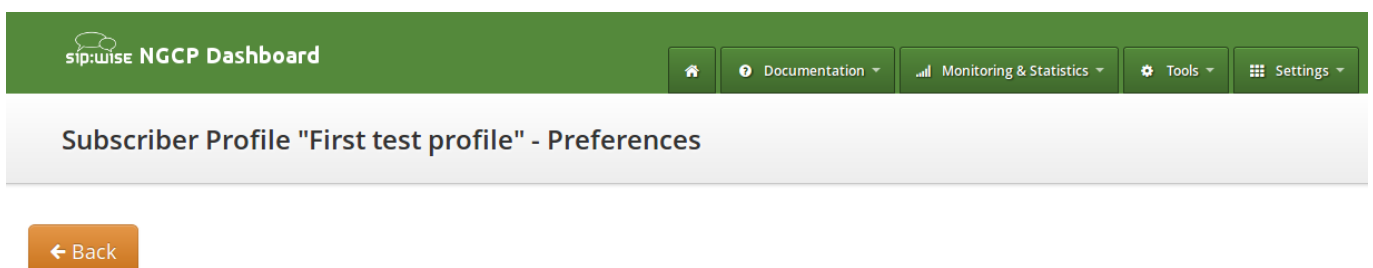
Checking the *Default Profile* option causes this profile to get assigned automatically to all subscribers, who have the profile set assigned. Other options define the user preferences which should be made available to the subscriber.

---

#### Note

When the platform administrator selects *Preferences* of the Subscriber Profile he will get an empty page like in the picture below, if none or only certain options are selected in the Subscriber Profile.

---



Some of the options, like `ncos` (NCOS level), will enable the definition of that preference within the Subscriber Profile Preferences. Thus all subscribers who have this profile assigned to will have the preference activated by default. The below picture shows the preferences linked to the sample Subscriber Profile:

### 5.18 SIP Loop Detection

In order to detect a SIP loop ( incoming call as a response for a call request ) sip:carrier checks the combination of *SIP-URI*, *To* and *From* headers.

This check can be enabled in config.yml by setting `kamailio.proxy.loop_detection.enable: 'yes'`. The system tolerates `kamailio.proxy.loop_detection.expire` seconds. Higher occurrence of loops will be reported with a SIP 482 "Loop Detected" error message

### 5.19 Call-Through Application

Call-through allows telephony client to dial into an IVR system and specify (in two-stage dialing fashion) a new destination number which is then dialed by the sip:carrier to connect the client to the destination. As the call-through system needs to be protected from unauthorized use, a list of CLIs which are allowed to use the call-through system is stored in the sip:carrier platform.

Table 6: Call-Through Mappings

Column	Description
uuid	The internal UUID of the call-through subscriber
auth_key	Authentication key (CLI)
source_uuid	The internal UUID of the subscriber that is authorized for outgoing call leg (same as uuid in call-through scenario)

## 5.19.1 Administrative Configuration

### 5.19.1.1 Subscriber provisioning

In order to manage the call-through CLIs for subscriber, navigate to *Settings*→*Subscribers*, search for the subscriber you want to edit, press *Details* and then *Preferences*, scroll down to the *Callthrough CLIs* section and press *Edit Callthrough CLIs* button.

Logged in as administrator

sip:wise NGCP Dashboard

Documentation Monitoring & Statistics

### Subscriber Preferences for 43993006@10.15.20.133

← Back

Successfully updated ccmappings

- Call Forwards
- Voicemail and Voicebox
- Fax Features
- Speed Dial
- Reminder
- Callthrough CLIs**

★ Edit Callthrough CLIs

Show 5 entries Search:

#	CLI	Source UUID
1	431234567890	8bb71bde-ab29-42cc-85dd-dff55e8b33d4

Showing 1 to 1 of 1 entries

Using the NGCP Panel the user then creates Call Forward to destination *Call Through*.

### 5.19.1.2 Forward to local user

If the subscriber has a Call Forward to the call-through application but caller's CLI is not in the authorized CLIs list for call-through, sems responds with error back to proxy and proxy advances to the next number in the Call Forward destinations set. User can enter special destination *Local Subscriber* as next target after *Call Through* in the destinations set in order to terminate the call to the subscriber as if the subscriber didn't exist. This way the user may reach the call-through application from his authorized CLI (e.g. mobile number) and all other callers would reach the SIP subscriber's registered phone as usual.



The screenshot shows the 'Edit Destination Set' dialog box in the NGCP Dashboard. The dialog has a green header with the title 'Edit Destination Set' and a close button. The main content area contains the following fields and options:

- Name:** quickset\_cfu
- Destination:** A list of radio buttons with 'Call Through' selected and highlighted by a red box. The other options are Voicemail, Conference, Fax2Mail, Calling Card, Local Subscriber, and URI/Number.
- URI/Number:** An empty text input field.
- for (seconds):** 300
- Priority:** 1
- Remove:** An orange button located to the right of the Priority field.

Below the main form, there is a second 'Destination' section with radio buttons. In this section, 'Local Subscriber' is selected and highlighted by a red box. The other options are Voicemail, Conference, Fax2Mail, Calling Card, and Call Through.

### 5.19.1.3 Sound Set provisioning

In order for the Callthrough application to work a Sound Set must be created and associated with the Domain or Subscriber.

Sound Sets can be defined in *Settings*→*Sound Sets*. To create a new Sound Set, click *Create Sound Set*. Then click the *Files* button. Administrator can upload the default sounds in one of supported languages or uploaded by the administrator manually in his language of choice.

There is a preference *sound\_set* on Domain and Subscriber levels to link subscribers to the sound set that they should hear (as usual the subscriber preference overrides the domain one).

sip:wise NGCP Dashboard

[🏠](#)
[📄 Documentation ▾](#)

## Manage Sound Set Calling Card and Call-through

← Back
★ Load Default Files

Sound set successfully loaded with default files.

calling\_card

Name	Filename	Loop	
and	and.wav	<input type="checkbox"/>	
busy_ringback_tone		<input type="checkbox"/>	
calling_card_not_found	calling_card_not_found.wav	<input type="checkbox"/>	
connecting	connecting.wav	<input type="checkbox"/>	
could_not_connect	could_not_connect.wav	<input type="checkbox"/>	

---

### Note

You may use 8 or 16 bit mono WAV audio files for all of the voice prompts.

---

## 5.19.2 Call Flow

The call arrives at sems application server with Request-URI user `callthrough`.

### 5.19.2.1 Internal Header Parameters

The INVITE contains an extra SIP header `P-App-Param` with the following parameters:

Table 7: SIP Header parameters for call-through application

Name	Meaning
uuid	The internal UUID of the call-through subscriber

Table 7: (continued)

Name	Meaning
srcnumber	Caller's CLI for the authentication
outgoing_cli	New CLI to be used by sems application for the outgoing call leg

### 5.19.2.2 Caller authorization

Caller is authorized using mapping shown in table above: `select source_uuid from provisioning.voip_cc_mapping where uuid=$uuid and auth_key=$srcnumber;`

If the check fails return the configured error response code. Then proceed with the call setup as follows.

### 5.19.2.3 Outgoing call

Sems requests the user to enter destination and starts digit collection. Digit collection process is terminated after 5 seconds (configurable in sems config file) or by pressing the # key. User can start entering destination while the voice prompt is being played.

Sems sends INVITE to the proxy with Request-URI: `sip:$number@$outboundproxy;sw_domain=$subscriber.domain`

From: `$outgoing_cli`

On receiving the 401 or 407 response from the proxy the application authenticates using the digest credentials retrieved for the call-through subscriber from the `voip_subscribers` table: `select s.username, s.password, d.domain from provisioning.voip_subscribers s, provisioning.voip_domains d where s.uuid=$source_uuid and s.domain_id=d.id;`

If the call setup fails the application plays back the "could\_not\_connect" sound file. If successful the application acts transparently and does not provide any voice announcements or DTMF detection.

### 5.19.2.4 CLI configuration

The CLI on the outgoing call from the call-through module is set to the Network-Provided Number (NPN) of the call-through subscriber. There is nothing to configure.

## 5.20 Calling Card Application

Calling card application uses a similar concept to call-through except that authorization process operates on the PIN code entered by user using DTMF instead of the CLI. The sip:carrier maps incoming UUID of the pilot subscriber to the list of PINs for calling card

application with their corresponding subscriber UUIDs for outbound call leg using table `provisioning.voip_cc_mapping`  
`table {"uuid", "auth_key", "source_uuid"}`

Table 8: Calling Cards

Column	Description
uuid	The internal UUID of the pilot subscriber
auth_key	Authentication key (PIN)
source_uuid	The internal UUID of the subscriber that is authorized for outgoing call leg

## 5.20.1 Administrative Configuration

### 5.20.1.1 Subscriber provisioning


In order to use the calling cards service the user creates a Call Forward to destination *Calling Card* for the designated subscriber that will be used as access number for this service.

### 5.20.1.2 Sound Set provisioning

In order for the Calling Card application to work a Sound Set must be created and associated with the Domain or Subscriber.

Sound Sets can be defined in *Settings*→*Sound Sets*. To create a new Sound Set, click *Create Sound Set*. Then click the *Files* button. Administrator can upload the default sounds in one of supported languages or uploaded by the administrator manually in his language of choice.

There is a preference *sound\_set* on Domain and Subscriber levels to link subscribers to the sound set that they should hear (as usual the subscriber preference overrides the domain one).



[🏠](#)
[📄 Documentation ▾](#)

## Manage Sound Set Calling Card and Call-through

← Back
★ Load Default Files

Sound set successfully loaded with default files.

calling\_card

Name	Filename	Loop	
and	and.wav	<input type="checkbox"/>	
busy_ringback_tone		<input type="checkbox"/>	
calling_card_not_found	calling_card_not_found.wav	<input type="checkbox"/>	
connecting	connecting.wav	<input type="checkbox"/>	
could_not_connect	could_not_connect.wav	<input type="checkbox"/>	

---

### Note

You may use 8 or 16 bit mono WAV audio files for all of the voice prompts.

---

### 5.20.1.3 CLI configuration

The CLI on the outgoing call from the calling card app can be configured in one of the following ways using subscriber preferences:

- 1) Show original caller's CLI: the calling card subscriber shall have `allowed_clis: * (any)`. Sems application sends the original caller's CLI in the From header, it is validated by the SIP proxy and sent to outside.
- 2) Show number of the pilot (calling card) subscriber: the calling card subscriber shall have an empty `allowed_clis` and desired number set as value of `user_cli` preference. The SIP proxy overrides the original caller's CLI in UPN with the value of the `user_cli` preference. The peer must have set `outbound_from_user, outbound_from_display: User-Provided Number (UPN)`.

## 5.20.2 Call Flow

The call arrives at sems application server with Request-URI user `callingcard`.

### 5.20.2.1 Internal Header Parameters

The INVITE contains an extra SIP header `P-App-Param` with the following parameters:

Table 9: SIP Header parameters for calling card application

Name	Meaning
<code>uuid</code>	The internal UUID of the pilot subscriber
<code>outgoing_cli</code>	New CLI to be used by sems application for the outgoing call leg

### 5.20.2.2 Caller authorization

- Sems requests the user to enter PIN and starts digit collection. Digit collection process is terminated after 5 seconds (configurable in sems config file) or by pressing the `#` key. User can start entering destination while the voice prompt is being played.
- Sems checks that PIN is valid and belongs to the pilot subscriber using mapping as shown in the table. It fetches UUID of the subscriber to be used for outgoing call leg: 

```
select source_uuid from provisioning.voip_cc_mapping where uuid=$uuid and auth_key=$pin;
```
- If the check fails sems will request the user to re-enter PIN up to the configured number of times.
- If successful proceed with the call setup making call on behalf of subscriber determined by the `source_uuid` key as follows.

### 5.20.2.3 Outgoing call

Sems application plays back the available balance of the customer. Sems requests the user to enter destination and starts digit collection. Digit collection process is terminated after 5 seconds (configurable in sems config file) or by pressing the `#` key. User can start entering destination while the voice prompt is being played.

Sems sends INVITE to the proxy with Request-URI: `sip:$number@$outboundproxy;sw_domain=$subscriber.domain`

From: `$outgoing_cli`

On receiving the 401 or 407 response from the proxy the application authenticates using the digest credentials retrieved for the subscriber for outgoing call leg from the `voip_subscribers` table: 

```
select s.username, s.password, d.domain from provisioning.voip_subscribers s, provisioning.voip_domains d where s.uuid=$source_uuid and s.domain_id=d.id;
```

#### 5.20.2.4 Voucher recharge

During the destination collection phase in calling card application user can enter special code \*1\*<pin># (configurable in sems config file) to transfer balance from other calling card customer to the currently authorized customer. Sems transfers all remaining balance from that customer to the current customer.

#### 5.20.2.5 Billing

The call via calling card application as well as call-through generates three CDRs:

- A to B: The incoming call from any source to the call-through subscriber.
- B to callingcard@app.local or callthrough@app.local: The call forward to the sems application.
- B to C: The outgoing call to the final destination. The three CDRs are handled by the billing process as usual, exported and shown in all call lists. .

### 5.21 Invoices and Invoice Templates

Content and vision of the invoices are customizable by [invoice templates](#) Section [5.21.3](#).

---

#### Note

The sip:carrier generates invoices in pdf format.

---

#### 5.21.1 Invoices Management

Invoices can be requested for generation, searched, downloaded and deleted on the administrative web interface. Navigate to *Settings* → *Invoices* menu and you get a list of all invoices currently stored in the database.

---

#### Tip

The system operator or a third party application can also generate, list, retrieve and delete invoices via the REST API. Please read further details [here](#) Section [5.21.2](#).

---

Logged in as administrator | Language | Logout

sip:wise NGCP Dashboard | Monitoring & Statistics | Settings

## Invoices

← Back | ★ Create Invoice

Show 5 entries | Search:

#	Customer #	Customer Email	Serial	
1	9	ipeshinskaya@sipwise.com	INV2014070000001	Download Delete
2	9	ipeshinskaya@sipwise.com	INV2014080000002	
4	143	ipeshinskaya@sipwise.com	INV2014080000004	

Showing 1 to 3 of 3 entries

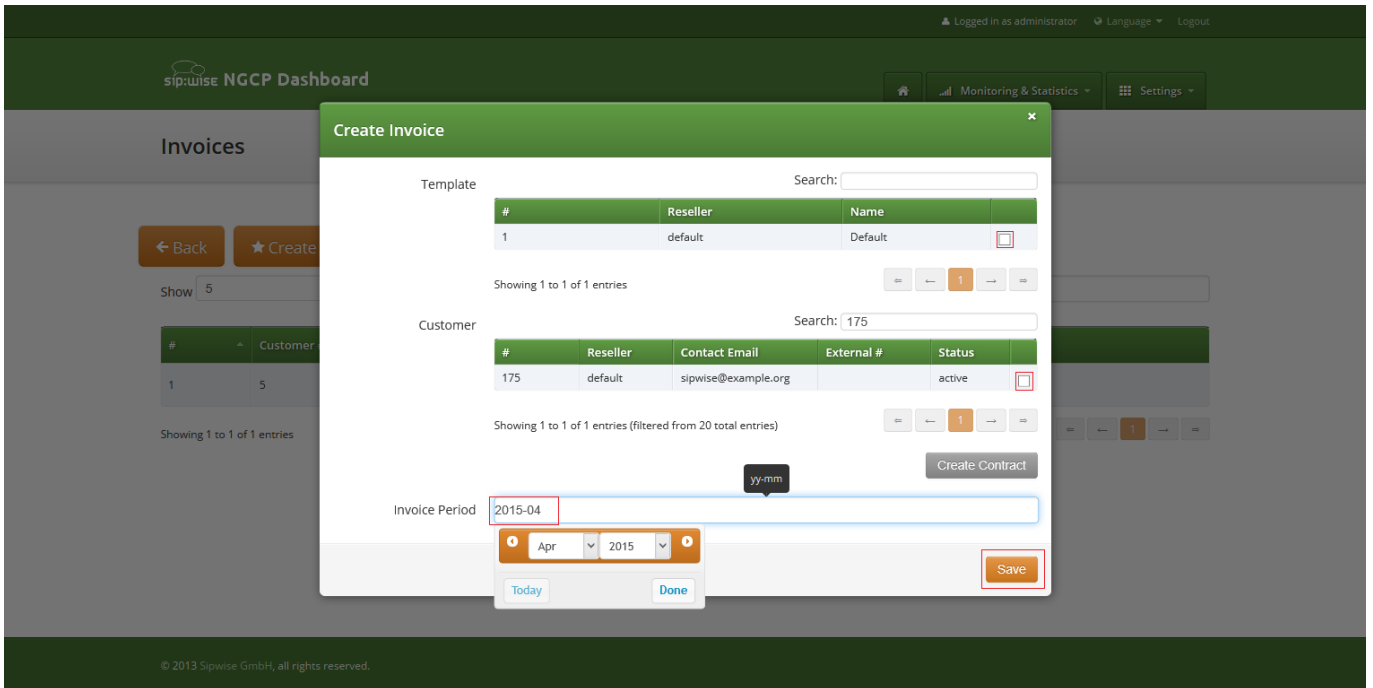
© 2013 Sipwise GmbH, all rights reserved.

To request invoice generation for the particular customer and period press "Create invoice" button. On the invoice creation form following parameters are available for selection:

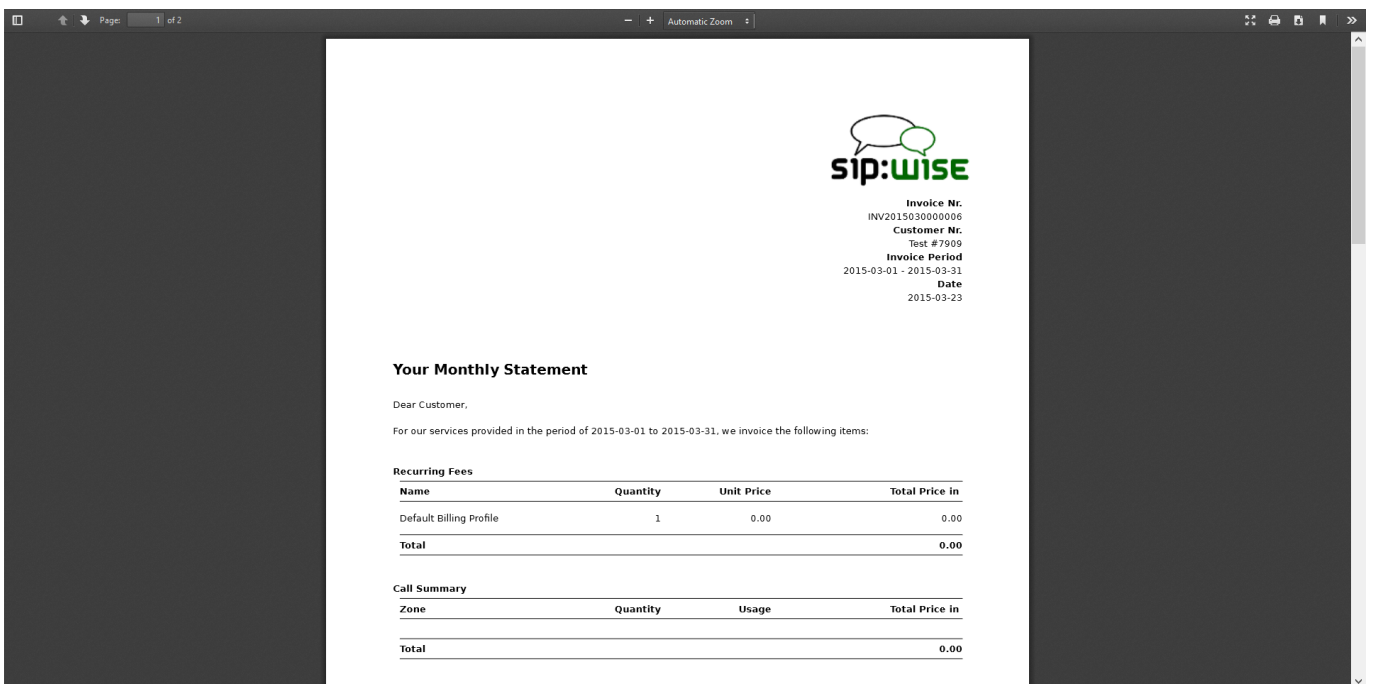
- **Template:** any of existent invoice template can be selected for the invoice generation.
- **Customer:** owner of the billing account, recipient of the invoice.
- **Invoice period:** billing period. Can be specified only as one calendar month. Calls with start time between first and last second of the period will be considered for the invoice

All form fields are mandatory.





Generated invoice can be downloaded as pdf file.



To do it press button "Download" against invoice in the invoice management interface.

Respectively press on the button "Delete" to delete invoice.

### 5.21.2 Invoice Management via REST API

Besides managing invoices on the admin web interface of NGCP, the system administrator (or a third party system) has the opportunity to request generation and retrieval of invoices via the *REST API*.

The subsequent sections describe the available operations for invoice management with API requests in details. All operations work on the *Invoices* resource and use the `/api/invoices` base path. The authentication method is username/password in the examples given below, however it is recommended to use a TLS client certificate for authentication on the REST API.

---

#### Note

The full API documentation is always available at the location: `https://<IP_of_NGCP_web_panel>:1443/api`

---

### 5.21.2.1 Generate a New Invoice

The **prerequisite** for generating a new invoice is that the customer has to have **an invoice template** assigned to him.

The following example shows a CURL command that will request generation of an invoice:

- for customer with ID "79"
- for the time period of November 2017
- based on the invoice template with ID "1"

```
curl -i -X POST -H 'Connection: close' -H 'Content-Type: application/json' \
  --user adminuser:adminpwd -k 'https://127.0.0.1:1443/api/invoices/' \
  --data-binary '{ "customer_id" : "79", "template_id" : "1", \
  "period_start": "2017-11-01 00:00:00", "period_end": "2017-11-30 23:59:59" }'
```

Please note that in this operation we used the `/api/invoices` path (the *invoices* collection) and a *POST* request on it to create a new invoice item.

In case of a **successful operation**, NGCP will reply with **201 Created** HTTP status and send the ID of the invoice in *Location* header. In our example the new invoice item may be directly referred as `/api/invoices/3` (ID = 3).

```
HTTP/1.1 201 Created
Server: nginx
Date: Tue, 14 Nov 2017 13:38:40 GMT
Content-Length: 0
Connection: close
Location: /api/invoices/3
Set-Cookie: ngcp_panel_session=d5e4a8dd003fd7cac646653a6b5aefa703cf3e66; path=/; expires= ↵
  Tue, 14-Nov-2017 14:38:38 GMT; HttpOnly
X-Catalyst: 5.90114
Strict-Transport-Security: max-age=15768000
```

In case of a **failed operation**, e.g. when we request an invoicing period that is invalid for the customer, NGCP will reply with **422 Unprocessable Entity** or **500 Internal Server Error** HTTP status.

### 5.21.2.2 Download Invoice Data

You can download properties / data of a specific invoice by selecting the item by its ID, using an HTTP *GET* request.

```
curl -i -X GET -H 'Connection: close' --user adminuser:adminpwd -k \  
'https://127.0.0.1:1443/api/invoices/3'
```

The above request will return a JSON data structure containing invoice properties:

```
HTTP/1.1 200 OK  
Server: nginx  
Date: Wed, 15 Nov 2017 12:13:04 GMT  
Content-Type: application/hal+json; profile="http://purl.org/sipwise/ngcp-api/"; charset= ↵  
utf-8  
Content-Length: 759  
Connection: close  
Link: </api/invoices/>; rel=collection  
Link: <http://purl.org/sipwise/ngcp-api/>; rel=profile  
Link: </api/invoices/3>; rel="item self"  
Link: </api/invoices/3>; rel="item http://purl.org/sipwise/ngcp-api/#rel-invoices"  
Link: </api/customers/79>; rel="item http://purl.org/sipwise/ngcp-api/#rel-customers"  
Set-Cookie: ngcp_panel_session=219fecbee4fa936defdlee511c84efe7b5a6d6a; path=/; expires= ↵  
Wed, 15-Nov-2017 13:13:03 GMT; HttpOnly  
Strict-Transport-Security: max-age=15768000  
  
{  
  "_links" : {  
    "collection" : {  
      "href" : "/api/invoices/"  
    },  
    "curies" : {  
      "href" : "http://purl.org/sipwise/ngcp-api/#rel-{rel}",  
      "name" : "ngcp",  
      "templated" : true  
    },  
    "ngcp:customers" : {  
      "href" : "/api/customers/79"  
    },  
    "ngcp:invoices" : {  
      "href" : "/api/invoices/3"  
    },  
    "profile" : {  
      "href" : "http://purl.org/sipwise/ngcp-api/"  
    },  
    "self" : {  
      "href" : "/api/invoices/3"  
    }  
  },  
  "amount_net" : 0,  
  "amount_total" : 0,  
  "amount_vat" : 0,  
}
```

```

    "id" : 3,
    "period_end" : "2017-11-30T23:59:59+00:00",
    "period_start" : "2017-11-01T00:00:00+00:00",
    "sent_date" : null,
    "serial" : "INV2017110000003"
}

```

It is also possible to query the complete *invoices* collection and use a filter (e.g. invoicing period, customer ID, etc.) to get the desired invoice item. In the example below we request all available invoices that belong to the customer with ID "79".

```

curl -i -X GET -H 'Connection: close' --user adminuser:adminpwd -k \
'https://127.0.0.1:1443/api/invoices/?customer_id=79'

```

The returned dataset is now slightly different because it is represented as an array of items, although in our example the array consist of only 1 item:

```

{
  "_embedded" : {
    "ngcp:invoices" : [
      {
        "_links" : {
          "collection" : {
            "href" : "/api/invoices/"
          },
          "curies" : {
            "href" : "http://purl.org/sipwise/ngcp-api/#rel-{rel}",
            "name" : "ngcp",
            "templated" : true
          },
          "ngcp:customers" : {
            "href" : "/api/customers/79"
          },
          "ngcp:invoices" : {
            "href" : "/api/invoices/3"
          },
          "profile" : {
            "href" : "http://purl.org/sipwise/ngcp-api/"
          },
          "self" : {
            "href" : "/api/invoices/3"
          }
        },
        "amount_net" : 0,
        "amount_total" : 0,
        "amount_vat" : 0,
        "id" : 3,
        "period_end" : "2017-11-30T23:59:59+00:00",
        "period_start" : "2017-11-01T00:00:00+00:00",

```

```

        "sent_date" : null,
        "serial" : "INV2017110000003"
    }
]
},
"_links" : {
    "curies" : {
        "href" : "http://purl.org/sipwise/ngcp-api/#rel-{rel}",
        "name" : "ngcp",
        "templated" : true
    },
    "ngcp:invoices" : {
        "href" : "/api/invoices/3"
    },
    "profile" : {
        "href" : "http://purl.org/sipwise/ngcp-api/"
    },
    "self" : {
        "href" : "/api/invoices/?page=1&rows=10"
    }
},
"total_count" : 1
}

```

### 5.21.2.3 Download Invoice as PDF File

You can download a specific invoice as a PDF file in the following way:

- selecting the item by its ID (as in our example, but you can also use a filter and query the complete *invoices* collection)
- using an HTTP *GET* request
- adding **"Accept: application/pdf"** header to the request

```

curl -X GET -H 'Connection: close' -H 'Accept: application/pdf' \
  --user adminuser:adminpwd -k 'https://127.0.0.1:1443/api/invoices/3' > result.pdf

```

**Please note** that in the example above we **do not add the "-i" option** that would also include the headers of the HTTP response in the output file. The output of the CURL command, i.e. the PDF file, is saved as "result.pdf" locally.

### 5.21.2.4 Delete an Invoice

In order to delete an invoice item you have to send a *DELETE* request on the specific item:

```

curl -i -X DELETE -H 'Connection: close' --user adminuser:adminpwd -k \
  'https://127.0.0.1:1443/api/invoices/3'

```

In case of successful deletion NGCP should send HTTP status 204 No Content as a response:

```
HTTP/1.1 204 No Content
Server: nginx
Date: Wed, 15 Nov 2017 13:42:42 GMT
Connection: close
Set-Cookie: ngcp_panel_session=10b66a6baf25a09739c2bb2377c70ecceee78387; path=/; expires= ↵
    Wed, 15-Nov-2017 14:42:42 GMT; HttpOnly
X-Catalyst: 5.90114
Strict-Transport-Security: max-age=15768000
```

### 5.21.3 Invoice Templates

Invoice template defines structure and look of the generated invoices. The sip:carrier makes it possible to create some invoice templates. Multiple invoice templates can be used to send invoices to the different customers using different languages.



#### Important

At least one invoice template should be created to enable invoice generation. Each customer has to be associated to one of the existent invoice template, otherwise invoices will be not generated for this customer.

Customer can be linked to the invoice template in the customer interface.

#### 5.21.3.1 Invoice Templates Management

Invoice templates can be searched, created, edited and deleted in the invoice templates management interface.

Logged in as administrator | Language | Logout

sip:wise NGCP Dashboard | Monitoring & Statistics | Settings

## Invoice Templates

← Back | ★ Create Invoice Template

Search:

#	Reseller	Name	Type	
1	default	Default	svg	<a href="#">Edit Meta</a> <a href="#">Edit Content</a> <a href="#">Delete</a>

Showing 1 to 1 of 1 entries

1

© 2013 Sipwise GmbH, all rights reserved.

Invoice template creation is separated on two steps:

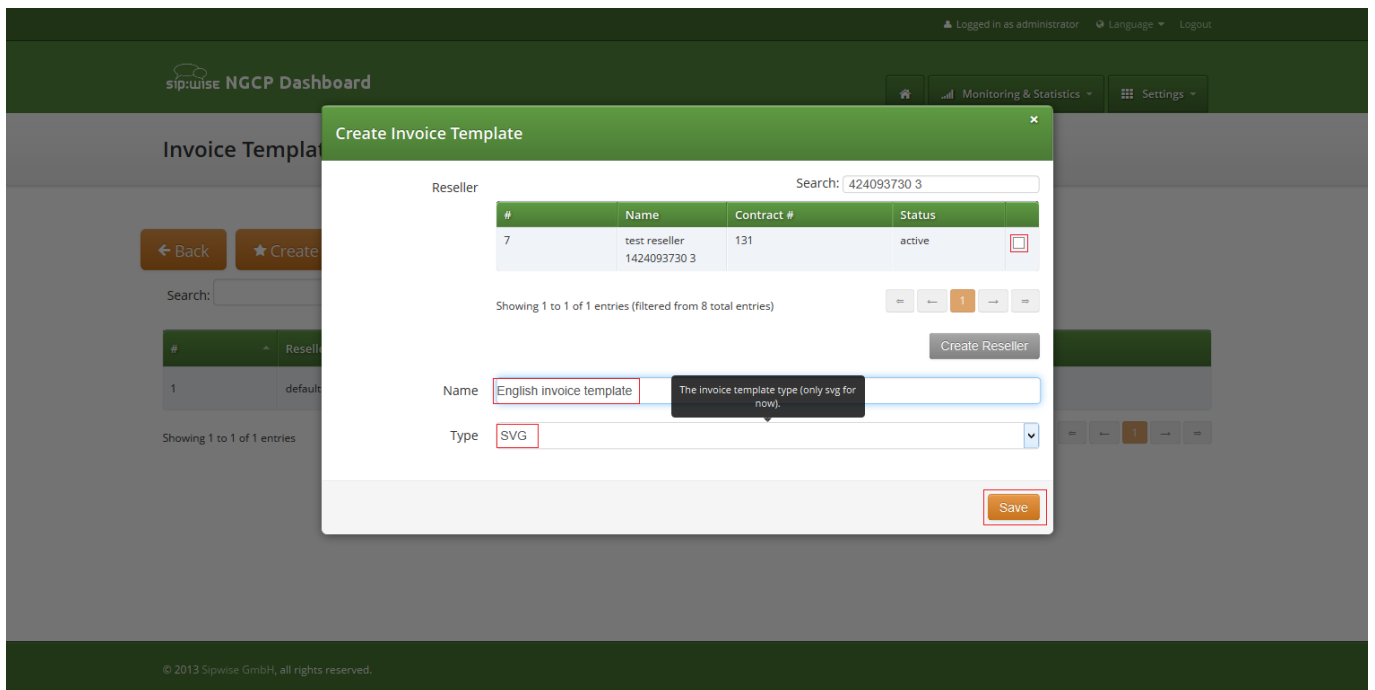
- Register new invoice template meta information.
- Edit content (template itself) of the invoice template.

To register new invoice template press "Create Invoice Template" button.

On the invoice template meta information form following parameters can be specified:

- **Reseller:** reseller who owns this invoice template. Please note, that it doesn't mean that the template will be used for the reseller customers by default. After creation, invoice template still need to be linked to the reseller customers.
- **Name:** unique invoice template name to differentiate invoice templates if there are some.
- **Type:** currently sip:carrier supports only svg format of the invoice templates.

All form fields are mandatory.



After registering new invoice template you can change invoice template structure in WYSIWYG SVG editor and preview result of the invoice generation based on the template.

### 5.21.3.2 Invoice Template Content

Invoice template is a XML SVG source, which describes content, look and position of the text lines, images or other invoice template elements. The sip:carrier provides embedded WYSIWYG SVG editor svg-edit 2.6 to customize default template. The sip:carrier svg-edit has some changes in layers management, image edit, user interface, but this [basic introduction](#) still may be useful.

Template refers to the owner reseller contact ("rescontact"), customer contract ("customer"), customer contact ("custcontact"), billing profile ("billprof"), invoice ("invoice") data as variables in the "[%%]" mark-up with detailed information accessed as field name after point e.g. [%invoice.serial%]. During invoice generation all variables or other special tokens in the "[% %]" mark-ups will be replaced by their database values.

Press on "Show variables" button on invoice template content page to see full list of variables with the fields:

You can add/change/remove embedded variables references directly in main svg-edit window. To edit text line in svg-edit main window double click on the text and place cursor on desired position in the text.

After implementation of the desired template changes, invoice template should be [saved](#) Section 5.21.3.3.

To return to the sip:carrier invoice template **default** content you can press on the "Discard changes" button.



### Important

"Discard changes" operation can't be undone.

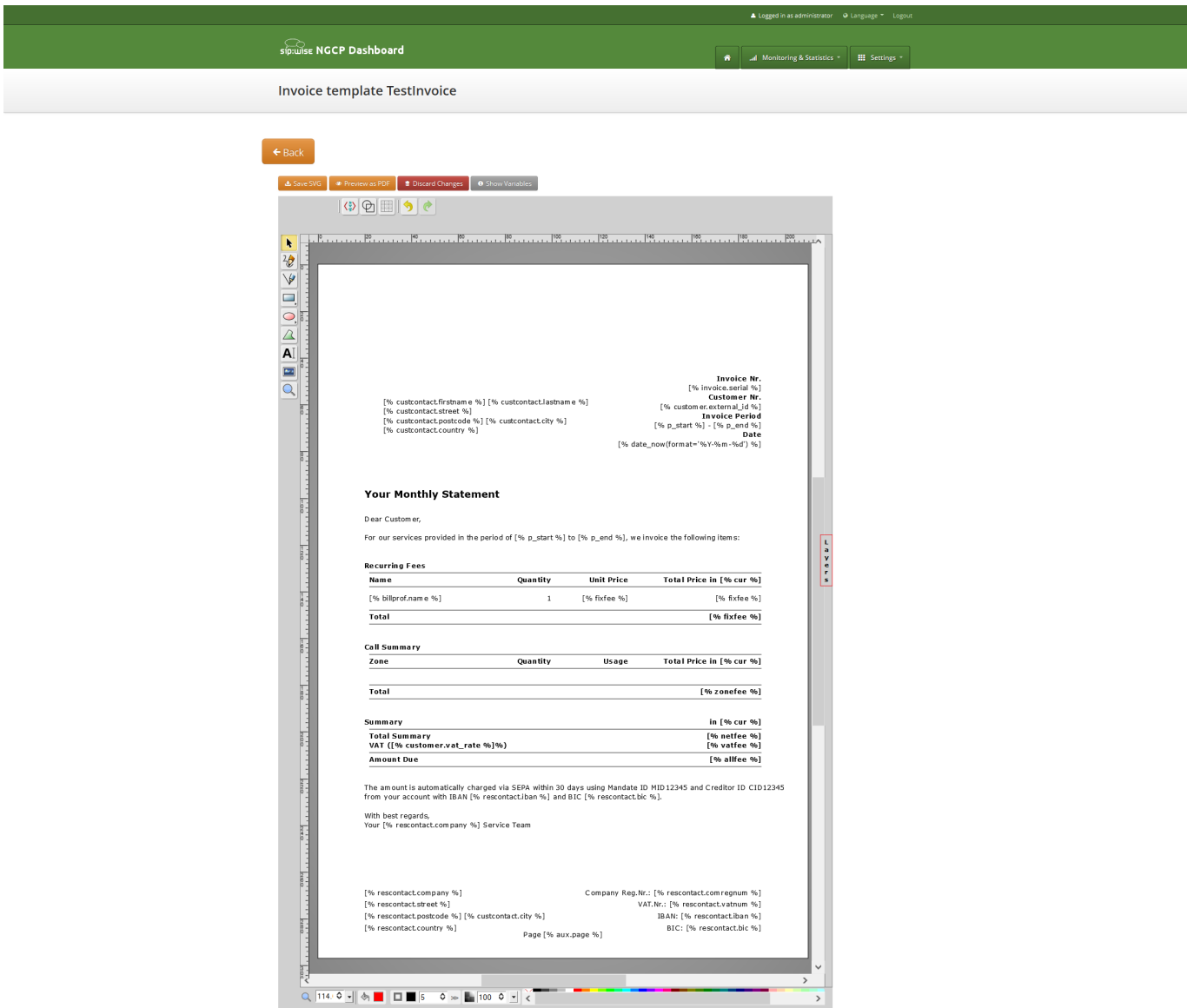
## Layers

Default template contains three groups elements (<g/>), which can be thought of as pages, or in terms of svg-edit - layers. Layers are:

- **Background:** special layer, which will be repeated as background for every other page of the invoice.
- **Summary:** page with a invoice summary.
- **CallList:** page with calls made in a invoice period. Is invisible by default.

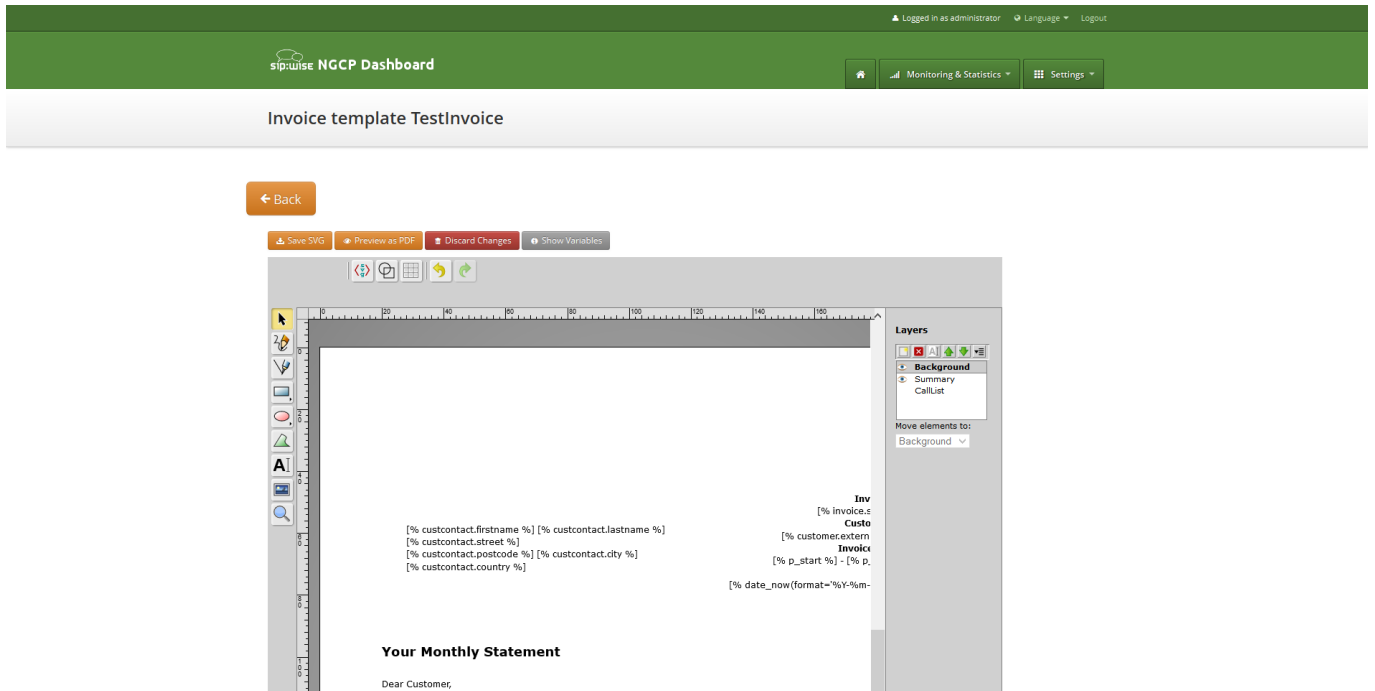


To see all invoice template layers, press on "Layers" vertical sign on right side of the svg-edit interface:



© 2013 Sipwise GmbH, all rights reserved.

Side panel with layers list will be shown.



One of the layers is active, and its element can be edited in the main svg-edit window. Currently active layer's name is **bold** in the layers list. The layers may be visible or invisible. Visible layers have "eye" icon left of their names in the layers list.

To make a layer active, click on its name in the layers list. If the layer was invisible, its elements became visible on activation. Thus you can see mixed elements of some layers, then you can switch off visibility of other layers by click on their "eye" icons. It is good idea to keep visibility of the "Background" layer on, so look of the generated page will be seen.

### **Edit SVG XML source**

Sometimes it may be convenient to edit svg source directly and svg-edit makes it possible to do it. After press on the <svg> icon in the top left corner of the svg-edit interface:

The screenshot shows the 'sip:wise NGCP Dashboard' interface. At the top, there is a navigation bar with 'Monitoring & Statistics' and 'Settings' menus. The main content area is titled 'Invoice template Rechnung\_v1'. Below this, there is a 'Back' button and a toolbar with 'Save SVG', 'Preview as PDF', 'Discard Changes', and 'Show Variables'. The editor displays a preview of an invoice template with placeholder text for customer details and invoice information.

SVG XML source of the invoice template will be shown.

SVG source can be edited in place or just copy-pasted as usual text.

---

### Note

Template keeps sizes and distances in pixels.

---

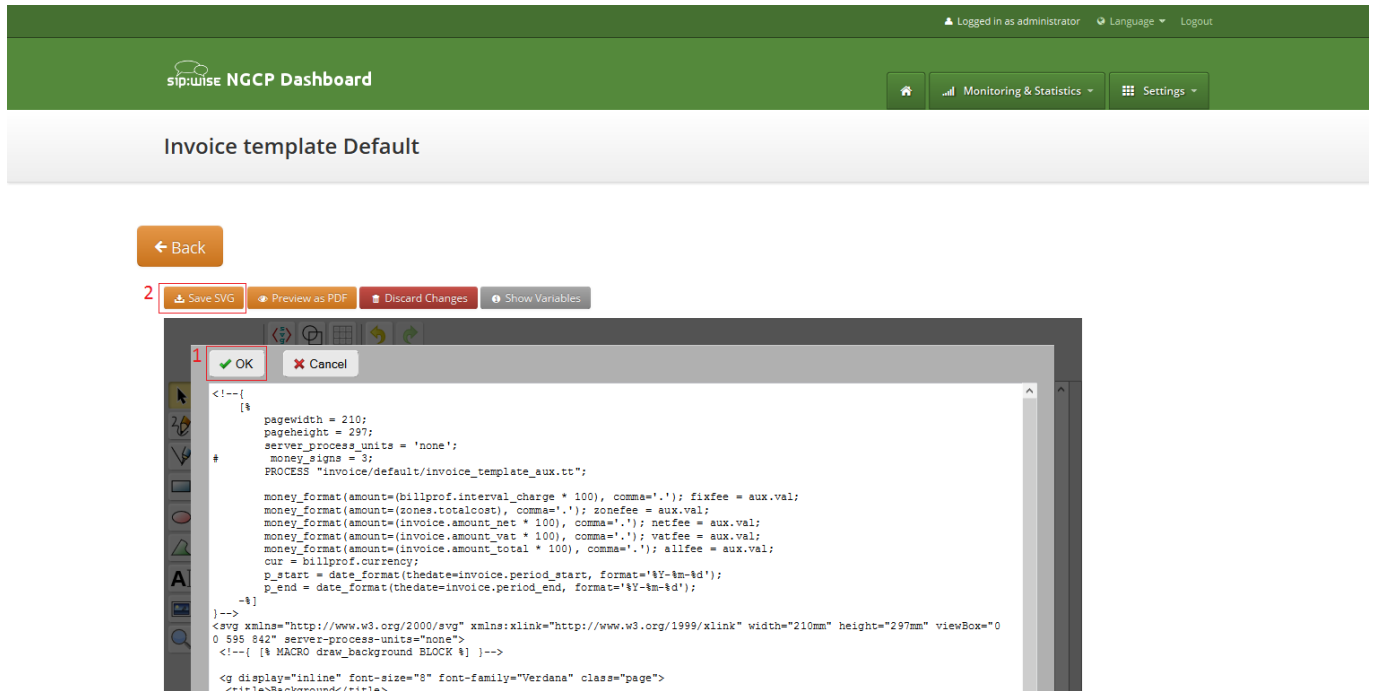


### Important

When edit svg xml source, please change very carefully and thoughtfully things inside special comment mark-up "`<!--{ }-->`". Otherwise invoice generation may be broken. Please be sure that document structure repeats default invoice template: has the same groups (`<g/>`) elements on the top level, text inside special comments mark-up "`<!--{ }-->`" preserved or changed appropriately, svg xml structure is correct.

---

To save your changes in the svg xml source, first press "OK" button on the top left corner of the source page:



The screenshot shows the 'Invoice template Default' configuration page in the sip:wise NGCP Dashboard. The dashboard header is green and contains the logo, 'Logged in as administrator', 'Language', and 'Logout'. Below the header, there are navigation buttons for 'Monitoring & Statistics' and 'Settings'. The main content area is white and features a 'Back' button and a toolbar with 'Save SVG', 'Preview as PDF', 'Discard Changes', and 'Show Variables'. A dialog box is open, displaying the SVG source code for the invoice template.

```

<!--{
  [%
    pagewidth = 210;
    pageheight = 297;
    server_process_units = 'none';
    money_signs = 3;
    PROCESS "invoice/default/invoice_template_aux.tt";
    money_format(amount=(billprof.interval_charge * 100), comma='.'); fixfee = aux.val;
    money_format(amount=(zones.totalcost), comma='.'); zonefee = aux.val;
    money_format(amount=(invoice.amount_net * 100), comma='.'); netfee = aux.val;
    money_format(amount=(invoice.amount_vat * 100), comma='.'); vatfee = aux.val;
    money_format(amount=(invoice.amount_total * 100), comma='.'); allfee = aux.val;
    aux = billprof.currency;
    p_start = date_format(thedate=invoice.period_start, format='%Y-%m-%d');
    p_end = date_format(thedate=invoice.period_end, format='%Y-%m-%d');
  -%]
-->
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="210mm" height="297mm" viewBox="0 0 595 842" server-process-units="none">
<!--[ [% MACRO draw_background BLOCK % ] ]-->
<g display="inline" font-size="8" font-family="Verdana" class="page">
<title>Background</title>

```

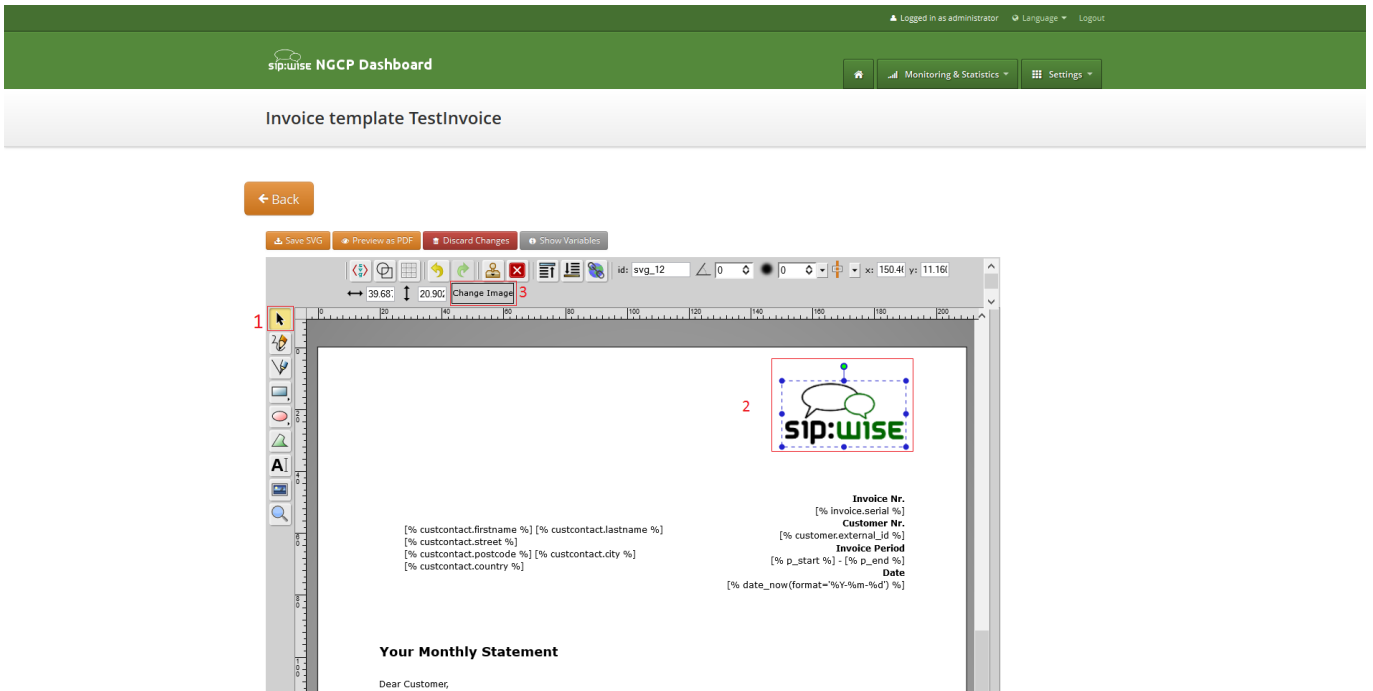
And then [save invoice template changes](#) Section 5.21.3.3.

### Note

You can copy and keep the svg source of your template as a file on the disk before start experimenting with the template. Later you will be able to return to this version replacing svg source.

### Change logo image

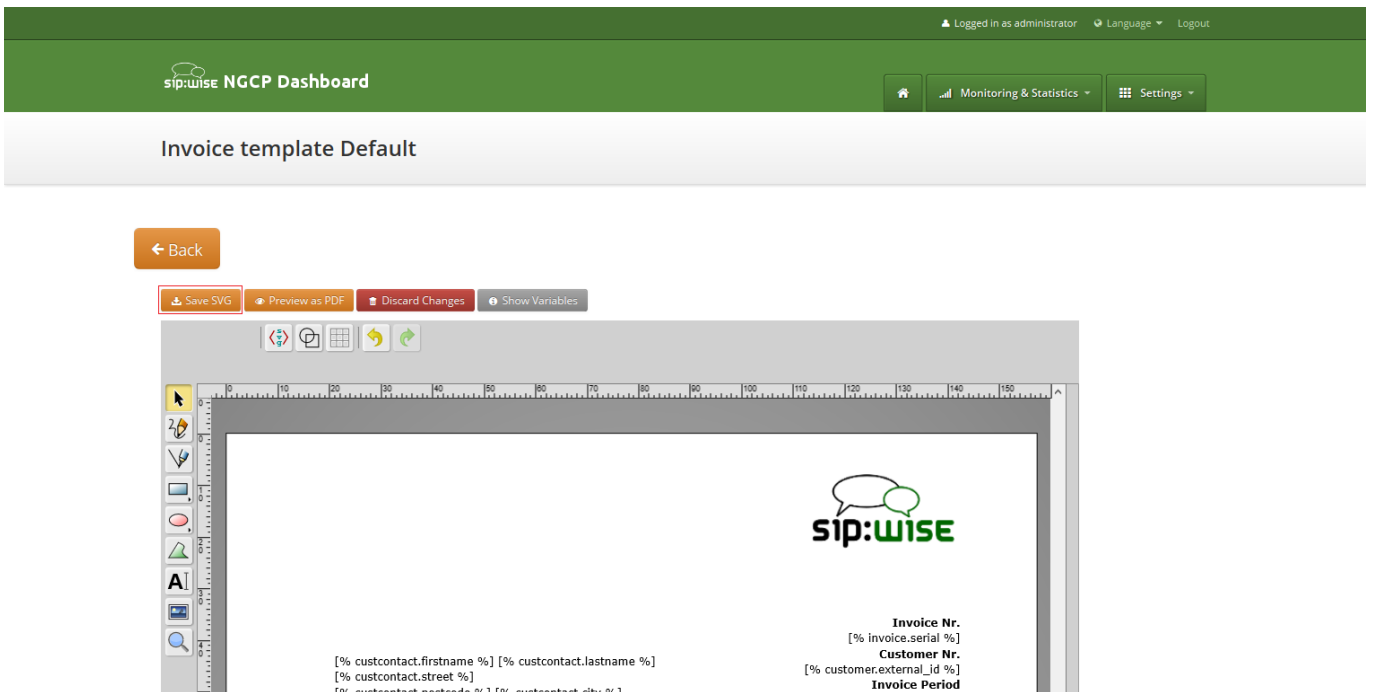
- Make sure that "Select tool" is active.
- Select default logo, clicking on the logo image.
- Press "Change image" button, which should appear on the top toolbar.



After image uploaded [save invoice template changes](#) Section 5.21.3.3.

### 5.21.3.3 Save and preview invoice template content

To save invoice template content changes press button "Save SVG".



You will see message about successfully saved template. You can preview your invoice look in PDF format. Press on "Preview as PDF" button.

Logged in as administrator | Language | Logout

**sip:wise NGCP Dashboard** | Home | Monitoring & Statistics | Settings

## Invoice template Default

[← Back](#)

Invoice template successfully saved



Invoice preview will be opened in the new window.

**Note**

Example fake data will be used for preview generation.

The screenshot shows a web browser window displaying a preview of an invoice. The invoice includes the sip:wise logo, customer information, invoice details, and a table of recurring fees.

**Customer Information:**  
 Customerfirst Customerlast  
 Customerstreet 12/3  
 12345 Customercity  
 Customercountry

**Invoice Details:**  
**Invoice Nr.** 1234567  
**Customer Nr.** Resext1234567890  
**Invoice Period** 2015-04-01 - 2015-04-30  
**Date** 2015-04-05

**Your Monthly Statement**

Dear Customer,

For our services provided in the period of 2015-04-01 to 2015-04-30, we invoice the following items:

Recurring Fees			
Name	Quantity	Unit Price	Total Price in EUR
Test Billing Profile	1	29.90	29.90
<b>Total</b>			<b>29.90</b>

## 5.22 Email Reports and Notifications

### 5.22.1 Email events

The sip:carrier makes it possible to customize content of the emails sent on the following actions:

- Web password reset requested. Email will be sent to the subscriber, whom password was requested for resetting. If the subscriber doesn't have own email, letter will be sent to the customer, who owns the subscriber.
- New subscriber created. Email will be sent to the newly created subscriber or to the customer, who owns new subscriber.
- Letter with the invoice. Letter will be sent to the customer.

### 5.22.2 Initial template values and template variables

Default email templates for each of the email events are inserted on the initial sip:carrier database creation. Content of the default template is described in the corresponding sections. Default email templates aren't linked to any reseller and can't be changed through sip:carrier Panel. They will be used to initialize default templates for the newly created reseller.

Each email template refers to the values from the database using special mark-ups "[%" and "%]". Each email template has fixed set of the variables. Variables can't be added or changed without changes in the sip:carrier Panel code.

### 5.22.3 Password reset email template

Email will be sent after subscriber or subscriber administrator requested password reset for the subscriber account. Letter will be sent to the subscriber. If subscriber doesn't have own email, letter will be sent to the customer owning the subscriber.

Default content of the password reset email template is:

<b>Template name</b>	<b>passreset_default_email</b>
<b>From</b>	default@sipwise.com
<b>Subject</b>	Password reset email
<b>Body</b>	<p>Dear Customer,</p> <p>Please go to [%url%] to set your password and log into your self-care ↔ interface.</p> <p>Your faithful Sipwise system</p> <p>--</p> <p>This is an automatically generated message. Do not reply.</p>

Following variables will be provided to the email template:

- [%url%]: specially generated url where subscriber can define his new password.
- [%subscriber%]: `username@domain` of the subscriber, which password was requested for reset.

#### 5.22.4 New subscriber notification email template

Email will be sent on the new subscriber creation. Letter will be sent to the newly created subscriber if it has an email. Otherwise, letter will be sent to the customer who owns the subscriber.

---

#### Note

By default email content template is addressed to the customer. Please consider this when create the subscriber with an email.

---

<b>Template name</b>	<b>subscriber_default_email</b>
<b>From</b>	<code>default@sipwise.com</code>
<b>Subject</b>	Subscriber created
<b>Body</b>	<pre> Dear Customer,  A new subscriber [%subscriber%] has been created for you.  Your faithful Sipwise system  --  This is an automatically generated message. Do not reply.</pre>

Following variables will be provided to the email template:

- [%url%]: specially generated url where subscriber can define his new password.
- [%subscriber%]: `username@domain` of the subscriber, which password was requested for reset.

#### 5.22.5 Invoice email template

<b>Template name</b>	<b>invoice_default_email</b>
<b>From</b>	<code>default@sipwise.com</code>
<b>Subject</b>	Invoice #[%invoice.serial%] from [%invoice.period_start_obj.ymd%] to [%invoice.period_end_obj.ymd%]



<b>Body</b>	<p>Dear Customer,</p> <p>Please find your invoice #[%invoice.serial%] for [%invoice. ←  period_start_obj.month_name%], [%invoice.period_start_obj.year%] in attachment  letter.</p> <p>Your faithful Sipwise system</p> <p>--</p> <p>This is an automatically generated message. Do not reply.</p>
-------------	--

Variables passed to the email template:

- [%**invoice**%]: container variable for the invoice information.

---

#### Invoice fields

- [%invoice.**serial**%]
- [%invoice.**amount\_net**%]
- [%invoice.**amount\_vat**%]
- [%invoice.**amount\_total**%]
- [%invoice.**period\_start\_obj**%]
- [%invoice.**period\_end\_obj**%]

The fields [%invoice.period\_start\_obj%] and [%invoice.period\_end\_obj%] provide methods of the perl package DateTime for the invoice start date and end date. Further information about DateTime can be obtained from the package documentation:  
man DateTime

---

- [%**provider**%]: container variable for the reseller contact. All database contact values will be available.
- [%**client**%]: container variable for the customer contact.

---

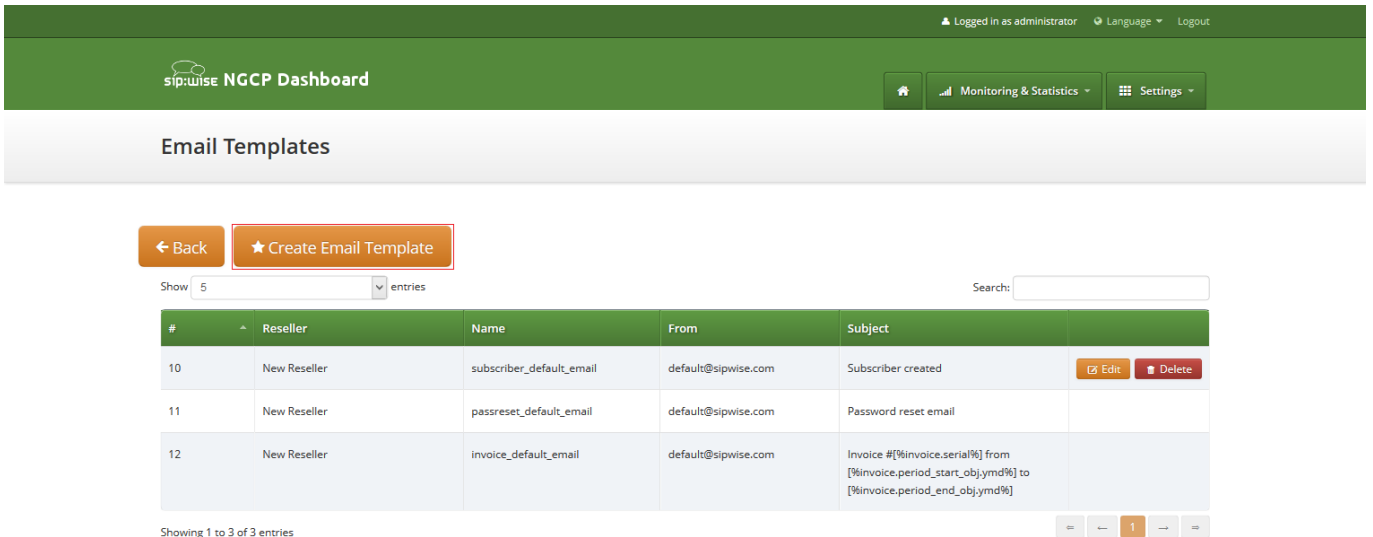
**Contact fields example for the "provider". Replace "provider" to client to access proper "customer" contact fields.**

- [%provider.gender%]
- [%provider.firstname%]
- [%provider.lastname%]
- [%provider.comregnum%]
- [%provider.company%]
- [%provider.street%]
- [%provider.postcode%]
- [%provider.city%]
- [%provider.country%]
- [%provider.phonenumber%]
- [%provider.mobilenumber%]
- [%provider.email%]
- [%provider.newsletter%]
- [%provider.faxnumber%]
- [%provider.iban%]
- [%provider.bic%]
- [%provider.vatnum%]
- [%provider.bankname%]
- [%provider.gpp0 - provider.gpp9%]

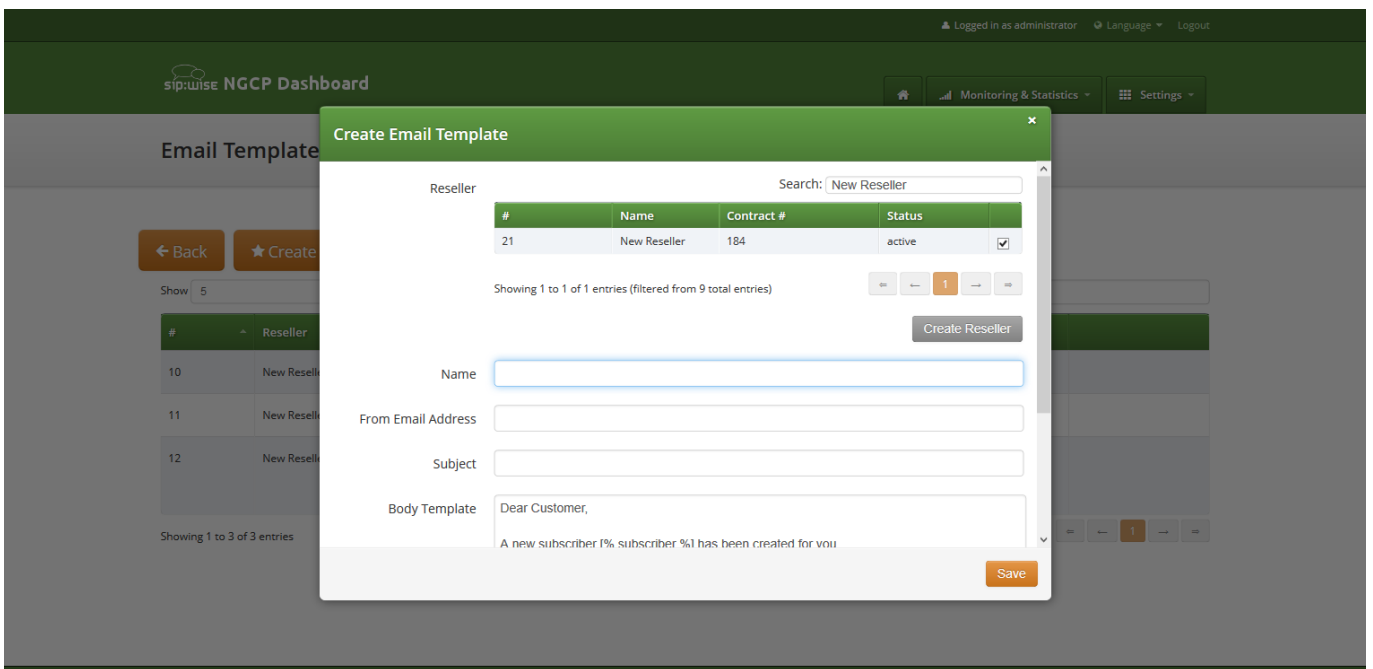
---

### 5.22.6 Email templates management

Email templates linked to the resellers can be customized in the email templates management interface. For the administrative account email templates of all the resellers will be shown. Respectively for the reseller account only owned email templates will be shown.



To create create new email template press button "Create Email Template".



On the email template form all fields are mandatory:

- **Reseller:** reseller who owns this email template.
- **Name:** currently only email template with the following names will be considered by the sip:carrier on the [appropriate event](#) Section 5.22.1 :
  - passreset\_default\_email;
  - subscriber\_default\_email;

- invoice\_default\_email;
- **From Email Address:** email address which will be used in the From field in the letter sent by the sip:carrier.
- **Subject:** Template of the email subject. Subject will be processed with the same template variables as the email body.
- **Body:** Email text template. Will be processed with appropriate template variables.

## 5.23 The Vertical Service Code Interface

*Vertical Service Codes (VSC)* are codes a user can dial on his phone to provision specific features for his subscriber account. The format is `*<code>*<value>` to activate a specific feature, and `#<code>` or `#<code>#` to deactivate it. The *code* parameter is a two-digit code, e.g. 72. The *value* parameter is the value being set for the corresponding feature.



### Important

The *value* user input is normalized using the Rewrite Rules Sets assigned to domain as described in Section 4.7.

---

By default, the following codes are configured for setting features. The examples below assume that there is a domain rewrite rule normalizing the number format `0<ac><sn>` to `<cc><ac><sn>` using 43 as country code.

- **72** - enable *Call Forward Unconditional* e.g. to 431000 by dialing `*72*01000`, and disable it by dialing `#72`.
- **90** - enable *Call Forward on Busy* e.g. to 431000 by dialing `*90*01000`, and disable it by dialing `#90`.
- **92** - enable *Call Forward on Timeout* e.g. after 30 seconds of ringing to 431000 by dialing `*92*30*01000`, and disable it by dialing `#92`.
- **93** - enable *Call Forward on Not Available* e.g. to 431000 by dialing `*93*01000`, and disable it by dialing `#93`.
- **50** - set *Speed Dial Slot*, e.g. set slot 1 to 431000 by dialing `*50*101000`, which then can be used by dialing `*1`.
- **55** - set *One-Shot Reminder Call* e.g. to 08:30 by dialing `*55*0830`.
- **31** - set *Calling Line Identification Restriction* for one call, e.g. to call 431000 anonymously dial `*31*01000`.
- **32** - enable *Block Incoming Anonymous Calls* by dialing `*32*`, and disable it by dialing `#32`.
- **80** - call using *Call Block Override PIN*, number should be prefixed with a block override PIN configured in admin panel to disable the outgoing user/admin block list and NCOS level for a call. For example, when override PIN is set to 7890, dial `*80*789001000` to call 431000 bypassing block lists.

### 5.23.1 Vertical Service Codes for PBX customers

Subscribers under the same PBX customer can enjoy some PBX-specific features by means of special VSCs.

NGCP provides the following PBX-specific VSCs:

- **97 - Call Parking:** during a conversation the subscriber can park the call with his phone to a "parking slot" and later on continue the conversation from another phone. To do that, a destination must be dialled as follows: \*97\*3; this will park the call to slot no. 3.

**PLEASE NOTE:**

- Cisco IP phones provide a softkey for Call Parking, that means the subscriber must only dial the parking slot number after pressing "Park" softkey on the phone.
  - Other IP phones can perform Call Parking as a *blind transfer*, where the destination of the transfer must be dialled in the format described above.
  - Both the caller and the callee can park the call.
- **98 - Call Unparking:** if a call has been parked, a subscriber may continue the conversation from any extension (phone) under the same PBX customer. To do that, the subscriber must dial the following sequence: \*98\*3; this will pick up the call that was parked at slot no. 3.
  - **99 - Directed Call Pickup:** if a subscriber's phone is ringing (e.g. extension 23) and another subscriber wants to answer the call instead of the original callee, he may pick up the call by dialling \*99\*23 on his phone.

### 5.23.2 Configuration of Vertical Service Codes

You can change any of the codes (but not the format) in `/etc/ngcp-config/config.yml` in the section `sems→vsc`. After the changes, execute `ngcpcfg apply "changed VSC codes"`.



**Caution**

If you have the EMTAs under your control, make sure that the specified VSCs don't overlap with EMTA-internal VSCs, because the VSC calls must be sent to the NGCP via SIP like normal telephone calls.

### 5.23.3 Voice Prompts for Vertical Service Code Configuration

Table 10: VSC Voice Prompts

Prompt Handle	Related VSC	Message
vsc_error	any	An error has occurred. Please try again later.
vsc_invalid	wrong code	Invalid feature code.
reject_vsc	any	Vertical service codes are disabled for this line.
vsc_cfu_on	72 (Call Forward Unconditional)	Your unconditional call forward has successfully been activated.
vsc_cfu_off	72 (Call Forward Unconditional)	Your unconditional call forward has successfully been deactivated.
vsc_cfb_on	90 (Call Forward Busy)	Your call forward on busy has successfully been activated.

Table 10: (continued)

Prompt Handle	Related VSC	Message
vsc_cfb_off	90 (Call Forward Busy)	Your call forward on busy has successfully been deactivated.
vsc_cft_on	92 (Call Forward on Timeout)	Your call forward on ring timeout has successfully been activated.
vsc_cft_off	92 (Call Forward on Timeout)	Your call forward on ring timeout has successfully been deactivated.
vsc_cfna_on	93 (Call Forward on Not Available)	Your call forward while not reachable has successfully been activated.
vsc_cfna_off	93 (Call Forward on Not Available)	Your call forward while not reachable has successfully been deactivated.
vsc_speeddial	50 (Speed Dial Slot)	Your speed dial slot has successfully been stored.
vsc_reminder_on	55 (One-Shot Reminder Call)	Your reminder has successfully been activated.
vsc_reminder_off	55 (One-Shot Reminder Call)	Your reminder has successfully been deactivated.
vsc_blockinclr_on	32 (Block Incoming Anonymous Calls)	Your rejection of anonymous calls has successfully been activated.
vsc_blockinclr_off	32 (Block Incoming Anonymous Calls)	Your rejection of anonymous calls has successfully been deactivated.

## 5.24 Handling WebRTC Clients

WebRTC is an open project providing browsers and mobile applications with Real-Time Communications (RTC) capabilities. Configuring your platform to offer WebRTC is quite easy and straightforward. This allows you to have a SIP-WebRTC bridge in place and make audio/video call towards normal SIP users from WebRTC clients and vice versa. Sip Provider listens, by default, on the following WebSockets and WebSocket Secure: `ws://your-ip:5060/ws`, `wss://your-ip:5061/ws` and `wss://your-ip:1443/wss/sip/`.

The WebRTC subscriber is just a normal subscriber which has just a different configuration in his Preferences. You need to change the following preferences under *Subscribers*→*Details*→*Preferences*→*NAT and Media Flow Control*:

- **use\_rtpproxy**: Always with rtpproxy as additional ICE candidate
- **transport\_protocol**: RTP/SAVPF (encrypted SRTP with RTCP feedback)

The `transport_protocol` setting may change, depending on your WebRTC client/browser configuration. Supported protocols are the following:

- Transparent (Pass through using the client's transport protocol)
- RTP/AVP (Plain RTP)
- RTP/SAVP (encrypted SRTP)
- RTP/AVPF (RTP with RTCP feedback)
- RTP/SAVPF (encrypted SRTP with RTCP feedback)
- UDP/TLS/RTP/SAVP (Encrypted SRTP using DTLS)
- UDP/TLS/RTP/SAVPF (Encrypted SRTP using DTLS with RTCP feedback)

**Warning**

The below configuration is enough to handle a WebRTC client/browser. As mentioned, you may need to tune a little bit your `transport_protocol` configuration, depending on your client/browser settings.

---

In order to have a bridge between normal SIP clients (using plain RTP for example) and WebRTC client, the normal SIP clients' preferences have to have the following configuration:

**transport\_protocol:** RTP/AVP (Plain RTP)

This will teach Sip Provider to translate between Plain RTP and RTP/SAVPF when you have calls between normal SIP clients and WebRTC clients.

## 5.25 XMPP and Instant Messaging

Instant Messaging (IM) based on XMPP comes with sip:carrier out of the box. sip:carrier uses `prosody` as internal XMPP server. Each subscriber created on the platform have assigned a XMPP user, reachable already - out of the box - by using the same SIP credentials. You can easily open an XMPP client (e.g. Pidgin) and login with your SIP `username@domain` and your SIP `password`. Then, using the XMPP client options, you can create your buddy list by adding your buddies in the format `user@domain`.

## 5.26 Call Recording

### 5.26.1 Introduction to Call Recording Function

Sipwise NGCP provides an opportunity to record call media content and store that in files. This function is available since mr5.3.1 version of the sip:carrier.

**Some characteristics of the Call Recording:**

- Call Recording function can store both unidirectional (originating either from caller, or from callee) or bidirectional (combined) streams from calls, resulting in 1, 2 or 3 physical files as output

- The location and format of the files is configurable.
- File storage is planned to occur on an NFS shared folder.
- Activation of call recording may happen generally for a *Domain / Peer / Subscriber* through the NGCP admin web interface.

**Important**

NGCP's Call Recording function is not meant for individual call interception purpose! Sipwise provides its Lawful Interception solution for that use case.

---

- Querying or deletion of existing recordings may happen through the REST API.
- Listing recordings of a subscriber is possible on NGCP's admin web interface.

The Call Recording function is implemented using NGCP's *rtengine* module.

**Note**

There are 2 *rtengine* daemons employed when call recording is enabled and active. The *main rtengine* takes care of forwarding media packets between caller and callee, as usual, while the *secondary rtengine* (recording) daemon is responsible for storing call data streams in the file system.

---

Call Recording is disabled by default. Enabling and configuration of Call Recording takes place in 2 steps:

1. Enabling the feature on the NGCP by setting configuration parameters in the main `config.yml` configuration file.
2. Activating the feature for a *Domain / Peer / Subscriber*.

**5.26.2 Information on Files and Directories**

NGCP's Call Recording function uses an **NFS shared folder** to save recorded streams.

**Important**

Since call data amount may be huge (depending, of course, on the number and duration of calls), it is *strongly not recommended* to store recorded streams on NGCP's local disks. However if you *have to* store recorded streams as files in the local filesystem, please contact Sipwise Support team in order to get the proper configuration of Call Recording function.

---

The NFS share gets mounted during startup of the recording daemon. If the NFS share cannot be mounted for some reason, the recording daemon will not start.

**Filenames** have the format: `<call_ID>-<random>-<SSRC>.<extension>`, where:

- `call_ID`: SIP Call-ID of the call being recorded
-



- `random`: is a string of random characters, unique for each recorded call. It's purpose is to avoid possible filename collisions if a Call-ID ever gets reused.
- `SSRC`: is the RTP SSRC for unidirectional recordings, or "mix" for the bidirectional (combined) audio.
- `extension`: is either "mp3" or "wav", depending on the configuration (`rtpproxy.recording.output_format`)

There might be 1, 2 or 3 files produced as recorded streams. The **number of files** depends on the configuration:

1. `rtpproxy.recording.output_mixed = 'yes'` (combined stream required)  
`rtpproxy.recording.output_single = 'no'` (unidirectional streams not required)
2. `rtpproxy.recording.output_mixed = 'no'` (combined stream not required)  
`rtpproxy.recording.output_single = 'yes'` (unidirectional streams required)
3. `rtpproxy.recording.output_mixed = 'yes'` (combined stream required)  
`rtpproxy.recording.output_single = 'yes'` (unidirectional streams required)

### 5.26.3 Configuration

The Call Recording function can be enabled and configured on the NGCP by changing the following configuration parameters in `config.yml` file:

```
rtpproxy:
  ...
  recording:
    enabled: no
    mp3_bitrate: '48000'
    nfs_host: 192.168.1.1
    nfs_remote_path: /var/recordings
    output_dir: /var/lib/rtpengine-recording
    output_format: wav
    output_mixed: yes
    output_single: yes
    resample: no
    resample_to: '16000'
    spool_dir: /var/spool/rtpengine
```

#### 5.26.3.1 Enabling Call Recording

Enabling the function requires changing the value of `rtpproxy.recording.enabled` parameter to "yes". In order to make the new configuration active, it's necessary to do:

```
ngcpcfg apply 'Activated call recording'
```

**Description of configuration parameters:**

- `enabled`: when set to "yes" Call Recording function is enabled; default: "no"
- `mp3_bitrate`: the bitrate used when recording happens in MP3 format; default: "48000"
- `nfs_host`: IP address of the NFS host that provides storage space for recorded streams; default: "192.168.1.1"
- `nfs_remote_path`: the remote path (folder) where files of recorded streams are stored on the NFS share; default: "/var/recordings"
- `output_dir`: is the local mount point for the NFS share, and thus where the final audio files will be written; default: "/var/lib/rtpengine-recording"

**Caution**

Normally you don't need to change the default setting. If you do change the value, please be aware that recorded files will be written by `root` user in that directory.

---

- `output_format`: possible values are "wav" (Wave) or "mp3" (MP3); default: "wav"
- `output_mixed`: "yes" means that there is a file that contains a mixed stream of caller and callee voice data; default: "yes"
- `output_single`: "yes" means that there is a separate file for each stream direction, i.e. for the streams originating from caller and callee; default: "yes"
- `resample`: when set to "yes" the call data stream will be resampled before storing it in the file; default: "no"
- `resample_to`: the sample rate used for resampling output; default: "16000"
- `spool_dir`: is the place for temporary metadata files that are used by the recording daemon and the main rtpengine daemon for their communication; default: "/var/spool/rtpengine"

**Caution**

You should not change the default setting unless you have a good reason to do so! Sipwise has thoroughly tested the Call Recording function with the default setting.

---

If Call Recording is enabled you can see 2 `rtpengine` processes running when checking the NGCP system state with `monit` tool:

```
root@sp1:/etc/ngcp-config# monit summary
...
Process 'lb'                               Running
Process 'rtpengine'                         Running
Process 'rtpengine-recording'              Running
Process 'voisniff-ng'                       Running
...
```

### 5.26.3.2 Activating Call Recording

Activating Call Recording for e.g. a *Subscriber*: please use NGCP's admin web interface for this purpose. On the web interface one has to navigate as follows: *Settings* → *Subscribers* → *select subscriber Details* → *Preferences* → *NAT and Media Flow Control*. Afterwards the `record_call` option has to be enabled by pressing the *Edit* button and ticking the checkbox.

NAT and Media Flow Control				
	Attribute	Name	Value	
	sound_set	System Sound Set	<input type="text"/>	
	use_rtpproxy	RTP-Proxy Mode	use domain default	
	ipv46_for_rtpproxy	IPv4/IPv6 bridging mode	use domain default	
	contract_sound_set	Customer Sound Set	<input type="text"/>	
	music_on_hold	Music on Hold	<input type="checkbox"/>	
	bypass_rtpproxy	Disable RTP-Proxy in the selected case	use domain default	
	rtp_interface	RTP interface	default	
	transport_protocol	Media transport protocol	use domain default	
	set_moh_sendonly	MoH sendonly	<input type="checkbox"/>	
	codecs_filter	Codecs filter	<input type="checkbox"/>	
	codecs_list	Codecs list		
	record_call	Record calls	<input type="checkbox"/>	<input type="button" value="Edit"/>

Figure 58: Activating Call Recording

#### Note

The call recording function may be activated for a single *Subscriber*, a *Domain* and a *Peer* server in the same way: *Preferences* → *NAT and Media Flow Control* → *record\_call*. When activating call recording for a *Domain* or *Peer* this effectively activates the function for all subscribers that belong to the selected domain, and for all calls with a local endpoint going through the selected peer server, respectively.

It is possible to **list existing call recordings** of a *Subscriber* through the admin web interface of NGCP. In order to do so, please navigate to: *Settings* → *Subscribers* → *select subscriber Details* → *Call Recordings*

Subscriber 43993002@10.15.18.222

← Back | Preferences | Calls history | Customer | Expand Groups

Master Data

Groups

Voice Mails

Call Recordings

From Date:  To Date:  Search:

#	Time	Call-ID
1	2017-09-05 11:51:41.543	17d5b961-7613-4ba2-9bc4-fb8086e2ccdd

Showing 1 to 1 of 1 entries

Figure 59: Listing Call Recordings

If you select an item in the list, besides the main properties such as the time of call and the SIP Call-ID, you can retrieve the details of the related call (press the *Call Details* button), get the list of recorded files (press the *Recorded Files* button) or *Delete* the recorded call.

When selecting *Call Details* you will see the most important accounting data of the call. Furthermore you can see the SIP *Call Flow* or the complete *Call Details* if you press the respective buttons.

Call List for 43993002@10.15.18.222 ( )

← Back

Show all calls

Show 5 entries From Date:  To Date:  Search:

#	Caller	Callee	CLIR	Billing zone	Status	Start Time	Duration	Call-ID	Cost
5	43993002	43993003	0	All Destinations	ok	2017-09-05 11:51:47.855	0:00:10.437	17d5b961-7613-4ba2-9bc4-fb8086e2ccdd	0.00
Total							0:00:10.437		0.00

Showing 1 to 1 of 1 entries

Figure 60: Listing Call Details for a Recording

When navigating to *Recorded Files* of a call you will be presented with a list of files. For each file item:

- type of stream is shown, that can be either "mixed" (combined voice data), or "single" (voice data of caller or callee)

- file `format` is shown, that can be either "wav", or "mp3"
- you can download the file by pressing the *Play* button

#	Type	Format
1	mixed	wav
3	single	wav
5	single	wav

Showing 1 to 3 of 3 entries

Figure 61: Listing Files for a Recording

#### 5.26.4 REST API

The NGCP REST API provides methods for querying and deletion of existing recording data. The full documentation of the available API methods is available on the admin web interface of the NGCP, as usual.

The following API methods are provided for managing Call Recordings:

- CallRecordings:
  - Provides information about the calls recorded in the system; can also be used to delete a recording entry
  - accessible by the path: `/api/callrecordings` (collection) or `/api/callrecordings/id` (single item)
  - Supported HTTP methods: OPTIONS, GET, DELETE
- CallRecordingStreams:
  - Provides information about recorded streams, such as start time, end time, format, mixed/single type, etc.; can also be used to delete a recorded stream
  - accessible by the path: `/api/callrecordingstreams` (collection) or `/api/callrecordingstreams/id` (single item)
  - Supported HTTP methods: OPTIONS, GET, DELETE
- CallRecordingFiles:
  - Provides information about recorded streams, such as start time, end time, format, mixed/single type, etc.; additionally returns the file content too
  - accessible by the path: `/api/callrecordingfiles` (collection) or `/api/callrecordingfiles/id` (single item)
  - Supported HTTP methods: OPTIONS, GET

## 5.27 SMS (Short Message Service) on Sipwise NGCP

Starting with its mr5.0.1 release, Sipwise NGCP offers *short messaging service* to its local subscribers. The implementation is based on a widely used software module: *Kannel*, and it needs to interact with a mobile operator's SMSC in order to send and receive SMS for the local subscribers. The data exchange with SMSC uses *SMPP* (Short Message Peer-to-Peer) protocol.

### SMS directions:

- incoming / received: the destination of the SM is a local subscriber on the NGCP
- outgoing / sent: the SM is submitted by a local subscriber

---

### Note

The Sipwise NGCP behaves as a short message client towards the SMSC of a mobile operator. This means every outgoing SM will be forwarded to the SMSC, and every incoming SM will reach the NGCP through an SMSC.

---

The architecture of the SMS components of NGCP and their interactions to other elements is depicted below, on a *sip:carrier* system:

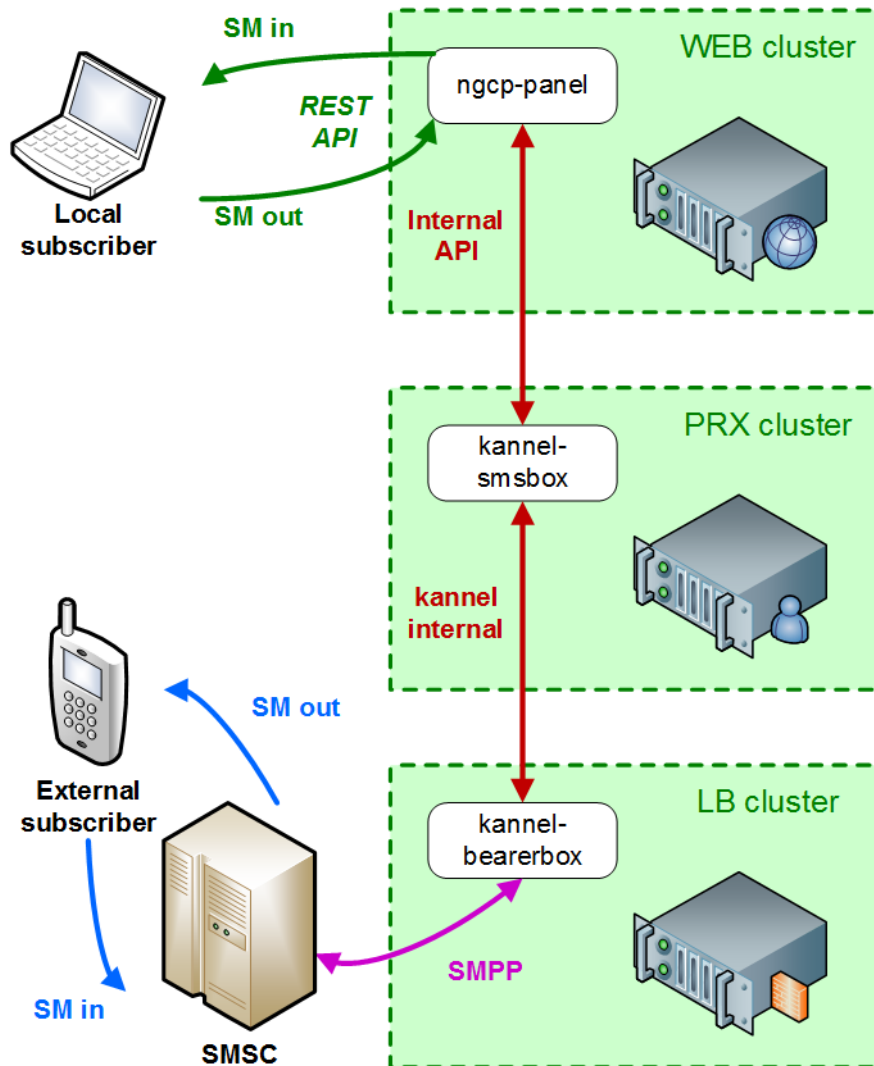


Figure 62: SMS Interactions on NGCP

**Note**

For the *sip:provider CE and PRO* NGCP installations: the *Kannel* components and the *ngcp-panel* all run on the same single node. The description of SMS module will continue referring to a *sip:carrier* installation in the handbook.

There are 2 components of the SMS module:

- **SMS Box:** this component takes care of handling the messages locally, that means:
  - delivering them to subscribers (writing into database for later retrieval)
  - picking up the submitted SMSs from the database and forwarding them to the *Bearer Box* component
- **Bearer Box:** this component manages the transmission of SMSs between Sipwise NGCP and the mobile operator's SMSC

## 5.27.1 Configuration

### 5.27.1.1 Main Parameters

The SMS function of NGCP is disabled by default. In order to **enable SMS** you have to change the value of configuration parameter `sms.enable` to `yes` in the main configuration file (`/etc/ngcp-config/config.yml`).

The second step of configuration is related to the **SMSC** where NGCP will connect to. You have to set the following parameters:

- `sms.smsc.host`: IP address of the SMSC
- `sms.smsc.port`: Port number of the SMSC
- `sms.smsc.username`: Username for authentication on the SMSC
- `sms.smsc.password`: Password for authentication on the SMSC

Other parameters of the SMSC connection may also need to be changed from the default values, but this is specific to each deployment.

Then, as usual, you have to make the new configuration active:

```
$ ngcpcfg apply 'Enabled SMS'
$ ngcpcfg push all
```

### 5.27.1.2 Configuration Files of Kannel

There are a few configuration files for the *Kannel* module, namely:

- `/etc/default/ngcp-kannel`: determines which components of *Kannel* will be started. This is auto-generated from `/etc/ngcp-config/templates/etc/default/ngcp-kannel.tt2` file when SMS is enabled.
- `/etc/kannel/kannel.conf`: contains detailed configuration of *Kannel* components. This is auto-generated from `/etc/ngcp-config/templates/etc/kannel/kannel.conf.tt2` file when SMS is enabled.
- `/etc/logrotate.d/ngcp-kannel.conf`: configuration of *logrotate* for *Kannel* log files. This is auto-generated from `/etc/ngcp-config/templates/etc/logrotate.d/ngcp-kannel.conf.tt2` file when SMS is enabled.



#### Caution

Please do not change settings in the above mentioned template files, unless you have to tailor *Kannel* settings to your specific needs!

---

Finally: see the description of each configuration parameter in the [appendix](#) Section [B.1.34](#).



### 5.27.1.3 Call Forwarding for SMS (CFS)

Any subscriber registered on NGCP can apply a call forwarding setting for short messages, referred to as "CFS" (Call Forward - SMS). If the CFS feature is enabled, he can receive the SMs on his mobile phone, for example, instead of retrieving the SMs through the REST API. This is much more convenient for users if they do not have an application on their smartphone or computer that could manage the SMs through the REST API.

In order to enable CFS you have to set the forwarding as usual on the admin web interface, or through the REST API. Navigate to *Subscribers* → *select one* → *Details* → *Preferences* → *Call Forwards* and press the *Edit* button.

Subscriber Preferences for 43993003@10.15.18.222

← Back Expand Groups

Successfully saved Call Forward

Call Forwards

Type	Answer Timeout	Destinations	Timeset	Sources	
Call Forward Unconditional					
Call Forward Busy					
Call Forward Timeout					
Call Forward Unavailable					
Call Forward SMS		435551234101@10.15.18.222	for 300s	always	all sources <span>Edit</span> <span>Delete</span>

Figure 63: Call Forward for SMS

## 5.27.2 Monitoring, troubleshooting

### 5.27.2.1 Bearer Box (LB node of NGCP)

On the LB node you can see a **process** named "**bearerbox**". This process has 2 **listening ports** assigned to it:

- 13000: this is the generic *Kannel* administration port, that belongs to the "core" component of Kannel.
- 13001: this is the communication port towards the *SMS Box* component running on PRX nodes of NGCP.

The *monit* tool also shows the *bearerbox* process in its status information:

```
$ monit summary
...
Process 'kannel-bearerbox'           Running
...
```

The following log files can provide information about the operation of *Bearer Box*:

- status messages and high level, short entries about sent and received messages: `/var/log/ngcp/kannel/kannel.log`

```
...
2017-09-26 08:57:32 [15922] [10] DEBUG: boxc_receiver: heartbeat with load value 0 ↔
    received
...
2017-09-26 11:12:06 [15922] [10] DEBUG: boxc_receiver: sms received
2017-09-26 11:12:06 [15922] [10] DEBUG: send_msg: sending msg to box: <192.168.1.4>
2017-09-26 11:12:06 [15922] [11] DEBUG: send_msg: sending msg to box: <192.168.1.4>
2017-09-26 11:12:06 [15922] [11] DEBUG: boxc_sender: sent message to <192.168.1.4>
2017-09-26 11:12:06 [15922] [10] DEBUG: boxc_receiver: got ack
...
```

- detailed information and message content of sent and received messages, link enquiries: `/var/log/kannel/smsc.log`

---

#### Note

Sent and received message examples shown here do not contain the full phone number and content for confidentiality reason.

---

#### – Example received message:

```
...
2017-09-26 12:09:36 [15922] [6] DEBUG: SMPP[default_smsc]: Got PDU:
2017-09-26 12:09:36 [15922] [6] DEBUG: SMPP PDU 0x7f2274025070 dump:
2017-09-26 12:09:36 [15922] [6] DEBUG:   type_name: deliver_sm
2017-09-26 12:09:36 [15922] [6] DEBUG:   command_id: 5 = 0x00000005
2017-09-26 12:09:36 [15922] [6] DEBUG:   command_status: 0 = 0x00000000
2017-09-26 12:09:36 [15922] [6] DEBUG:   sequence_number: 11867393 = 0x00b51501
2017-09-26 12:09:36 [15922] [6] DEBUG:   service_type: NULL
2017-09-26 12:09:36 [15922] [6] DEBUG:   source_addr_ton: 2 = 0x00000002
2017-09-26 12:09:36 [15922] [6] DEBUG:   source_addr_npi: 1 = 0x00000001
2017-09-26 12:09:36 [15922] [6] DEBUG:   source_addr: "0660....."
2017-09-26 12:09:36 [15922] [6] DEBUG:   dest_addr_ton: 1 = 0x00000001
2017-09-26 12:09:36 [15922] [6] DEBUG:   dest_addr_npi: 1 = 0x00000001
2017-09-26 12:09:36 [15922] [6] DEBUG:   destination_addr: "43668....."
2017-09-26 12:09:36 [15922] [6] DEBUG:   esm_class: 0 = 0x00000000
2017-09-26 12:09:36 [15922] [6] DEBUG:   protocol_id: 0 = 0x00000000
2017-09-26 12:09:36 [15922] [6] DEBUG:   priority_flag: 0 = 0x00000000
2017-09-26 12:09:36 [15922] [6] DEBUG:   schedule_delivery_time: NULL
2017-09-26 12:09:36 [15922] [6] DEBUG:   validity_period: NULL
2017-09-26 12:09:36 [15922] [6] DEBUG:   registered_delivery: 0 = 0x00000000
2017-09-26 12:09:36 [15922] [6] DEBUG:   replace_if_present_flag: 0 = 0x00000000
2017-09-26 12:09:36 [15922] [6] DEBUG:   data_coding: 3 = 0x00000003
2017-09-26 12:09:36 [15922] [6] DEBUG:   sm_default_msg_id: 0 = 0x00000000
2017-09-26 12:09:36 [15922] [6] DEBUG:   sm_length: 158 = 0x0000009e
```

```

2017-09-26 12:09:36 [15922] [6] DEBUG: short_message:
2017-09-26 12:09:36 [15922] [6] DEBUG:   Octet string at 0x7f2274000f80:
2017-09-26 12:09:36 [15922] [6] DEBUG:     len: 158
2017-09-26 12:09:36 [15922] [6] DEBUG:     size: 159
2017-09-26 12:09:36 [15922] [6] DEBUG:     immutable: 0
2017-09-26 12:09:36 [15922] [6] DEBUG:     data: 5a <14 bytes> 46
2017-09-26 12:09:36 [15922] [6] DEBUG:     data: 72 <14 bytes> 68
2017-09-26 12:09:36 [15922] [6] DEBUG:     data: 61 <14 bytes> 67
2017-09-26 12:09:36 [15922] [6] DEBUG:     data: 20 <14 bytes> 57
2017-09-26 12:09:36 [15922] [6] DEBUG:     data: 65 <14 bytes> 63
2017-09-26 12:09:36 [15922] [6] DEBUG:     data: 68 <14 bytes> 73
2017-09-26 12:09:36 [15922] [6] DEBUG:     data: 2e <14 bytes> 61
2017-09-26 12:09:36 [15922] [6] DEBUG:     data: 6c <14 bytes> 73
2017-09-26 12:09:36 [15922] [6] DEBUG:     data: 3a <14 bytes> 73
2017-09-26 12:09:36 [15922] [6] DEBUG:     data: 4d <14 bytes> 6e
2017-09-26 12:09:36 [15922] [6] DEBUG:   Octet string dump ends.
2017-09-26 12:09:36 [15922] [6] DEBUG: SMPP PDU dump ends.
2017-09-26 12:09:36 [15922] [6] DEBUG: SMPP[default_smsc]: Sending PDU:
2017-09-26 12:09:36 [15922] [6] DEBUG: SMPP PDU 0x7f2274020790 dump:
2017-09-26 12:09:36 [15922] [6] DEBUG:   type_name: deliver_sm_resp
2017-09-26 12:09:36 [15922] [6] DEBUG:   command_id: 2147483653 = 0x80000005
2017-09-26 12:09:36 [15922] [6] DEBUG:   command_status: 0 = 0x00000000
2017-09-26 12:09:36 [15922] [6] DEBUG:   sequence_number: 11867393 = 0x00b51501
2017-09-26 12:09:36 [15922] [6] DEBUG:   message_id: NULL
2017-09-26 12:09:36 [15922] [6] DEBUG: SMPP PDU dump ends.
2017-09-26 12:09:36 [15922] [6] DEBUG: SMPP[default_smsc]: throughput (0.00,5.00)
...

```

– Example sent message:

```

...
2017-09-26 12:04:08 [15922] [6] DEBUG: SMPP[default_smsc]: throughput (0.00,5.00)
2017-09-26 12:04:08 [15922] [6] DEBUG: SMPP[default_smsc]: Manually forced source addr ↔
   ton = 1, source add npi = 1
2017-09-26 12:04:08 [15922] [6] DEBUG: SMPP[default_smsc]: Manually forced dest addr ton ↔
   = 1, dest add npi = 1
2017-09-26 12:04:08 [15922] [6] DEBUG: SMPP[default_smsc]: Sending PDU:
2017-09-26 12:04:08 [15922] [6] DEBUG: SMPP PDU 0x7f2274025070 dump:
2017-09-26 12:04:08 [15922] [6] DEBUG:   type_name: submit_sm
2017-09-26 12:04:08 [15922] [6] DEBUG:   command_id: 4 = 0x00000004
2017-09-26 12:04:08 [15922] [6] DEBUG:   command_status: 0 = 0x00000000
2017-09-26 12:04:08 [15922] [6] DEBUG:   sequence_number: 98163 = 0x00017f73
2017-09-26 12:04:08 [15922] [6] DEBUG:   service_type: NULL
2017-09-26 12:04:08 [15922] [6] DEBUG:   source_addr_ton: 5 = 0x00000005
2017-09-26 12:04:08 [15922] [6] DEBUG:   source_addr_npi: 0 = 0x00000000
2017-09-26 12:04:08 [15922] [6] DEBUG:   source_addr: "any"
2017-09-26 12:04:08 [15922] [6] DEBUG:   dest_addr_ton: 1 = 0x00000001
2017-09-26 12:04:08 [15922] [6] DEBUG:   dest_addr_npi: 1 = 0x00000001

```

```

2017-09-26 12:04:08 [15922] [6] DEBUG: destination_addr: "43676....."
2017-09-26 12:04:08 [15922] [6] DEBUG: esm_class: 3 = 0x00000003
2017-09-26 12:04:08 [15922] [6] DEBUG: protocol_id: 0 = 0x00000000
2017-09-26 12:04:08 [15922] [6] DEBUG: priority_flag: 0 = 0x00000000
2017-09-26 12:04:08 [15922] [6] DEBUG: schedule_delivery_time: NULL
2017-09-26 12:04:08 [15922] [6] DEBUG: validity_period: NULL
2017-09-26 12:04:08 [15922] [6] DEBUG: registered_delivery: 0 = 0x00000000
2017-09-26 12:04:08 [15922] [6] DEBUG: replace_if_present_flag: 0 = 0x00000000
2017-09-26 12:04:08 [15922] [6] DEBUG: data_coding: 0 = 0x00000000
2017-09-26 12:04:08 [15922] [6] DEBUG: sm_default_msg_id: 0 = 0x00000000
2017-09-26 12:04:08 [15922] [6] DEBUG: sm_length: 23 = 0x00000017
2017-09-26 12:04:08 [15922] [6] DEBUG: short_message:
2017-09-26 12:04:08 [15922] [6] DEBUG:   Octet string at 0x7f227400c460:
2017-09-26 12:04:08 [15922] [6] DEBUG:     len: 23
2017-09-26 12:04:08 [15922] [6] DEBUG:     size: 24
2017-09-26 12:04:08 [15922] [6] DEBUG:     immutable: 0
2017-09-26 12:04:08 [15922] [6] DEBUG:     data: 44 <14 bytes> 73
2017-09-26 12:04:08 [15922] [6] DEBUG:     data: 74 <5 bytes> 39
2017-09-26 12:04:08 [15922] [6] DEBUG:   Octet string dump ends.
2017-09-26 12:04:08 [15922] [6] DEBUG: SMPP PDU dump ends.
2017-09-26 12:04:08 [15922] [6] DEBUG: SMPP[default_smsc]: throughput (1.00,5.00)
...

```

– Example link enquiry:

```

...
2017-09-26 12:13:38 [15922] [6] DEBUG: SMPP[default_smsc]: throughput (0.00,5.00)
2017-09-26 12:13:38 [15922] [6] DEBUG: SMPP[default_smsc]: Got PDU:
2017-09-26 12:13:38 [15922] [6] DEBUG: SMPP PDU 0x7f2274020790 dump:
2017-09-26 12:13:38 [15922] [6] DEBUG:   type_name: enquire_link
2017-09-26 12:13:38 [15922] [6] DEBUG:   command_id: 21 = 0x00000015
2017-09-26 12:13:38 [15922] [6] DEBUG:   command_status: 0 = 0x00000000
2017-09-26 12:13:38 [15922] [6] DEBUG:   sequence_number: 90764 = 0x0001628c
2017-09-26 12:13:38 [15922] [6] DEBUG: SMPP PDU dump ends.
2017-09-26 12:13:38 [15922] [6] DEBUG: SMPP[default_smsc]: Sending PDU:
2017-09-26 12:13:38 [15922] [6] DEBUG: SMPP PDU 0x7f2274025070 dump:
2017-09-26 12:13:38 [15922] [6] DEBUG:   type_name: enquire_link_resp
2017-09-26 12:13:38 [15922] [6] DEBUG:   command_id: 2147483669 = 0x80000015
2017-09-26 12:13:38 [15922] [6] DEBUG:   command_status: 0 = 0x00000000
2017-09-26 12:13:38 [15922] [6] DEBUG:   sequence_number: 90764 = 0x0001628c
2017-09-26 12:13:38 [15922] [6] DEBUG: SMPP PDU dump ends.
2017-09-26 12:13:38 [15922] [6] DEBUG: SMPP[default_smsc]: throughput (0.00,5.00)
...

```

### 5.27.2.2 SMS Box (PRX node of NGCP)

On the PRX node you can see a **process** named "**smsbox**". This process has a **listening port** assigned to it: 13002, that is the communication port towards the *Bearer Box* component running on LB nodes.

The *monit* tool also shows the *smsbox* process in its status information:

```
$ monit summary
...
Process 'kannel-smsbox'           Running
...
```

The following log files can provide information about the operation of *SMS Box*:

- sent and received messages using the API of WEB node: `/var/log/kannel/smsbox.log`

---

#### Note

Sent and received message examples shown here do not contain the full phone number and content for confidentiality reason.

---

#### – Example sent message:

```
...
2017-09-26 12:16:42 [22763] [2] DEBUG: HTTP: Creating HTTPClient for '192.168.1.2'.
2017-09-26 12:16:42 [22763] [2] DEBUG: HTTP: Created HTTPClient area 0x7f5dcc000ad0.
2017-09-26 12:16:42 [22763] [3] INFO: smsbox: Got HTTP request </cgi-bin/sendsms> from ↔
<192.168.1.3>
2017-09-26 12:16:42 [22763] [3] INFO: sendsms used by <sipwise>
2017-09-26 12:16:42 [22763] [3] INFO: sendsms sender:<sipwise:43668.....> ↔
(192.168.1.3) to:<43676.....> msg:<...>
2017-09-26 12:16:42 [22763] [3] DEBUG: Stored UUID ab95eb45-1ec0-4932-9863-1a95609a025f
2017-09-26 12:16:42 [22763] [3] DEBUG: message length 52, sending 1 messages
2017-09-26 12:16:42 [22763] [3] DEBUG: Status: 202 Answer: <Sent.>
2017-09-26 12:16:42 [22763] [3] DEBUG: Delayed reply - wait for bearerbox
2017-09-26 12:16:42 [22763] [0] DEBUG: Got ACK (0) of ab95eb45-1ec0-4932-9863-1 ↔
a95609a025f
2017-09-26 12:16:42 [22763] [0] DEBUG: HTTP: Destroying HTTPClient area 0x7f5dcc000ad0.
2017-09-26 12:16:42 [22763] [0] DEBUG: HTTP: Destroying HTTPClient for '192.168.1.3'.
...
```

#### – Example received message:

```
...
2017-09-26 11:59:45 [22763] [5] INFO: Starting to service <...message content...> from ↔
<+43676-----> to <+43668----->
2017-09-26 11:59:45 [22763] [10] DEBUG: Queue contains 0 pending requests.
2017-09-26 11:59:45 [22763] [10] DEBUG: HTTPS URL; Using SSL for the connection
2017-09-26 11:59:45 [22763] [10] DEBUG: Parsing URL 'https://192.168.1.2:1443/ ↔
internalsms/receive?auth_token=fNLosMgwdNUrKvEfFMm9
```

```

&timestamp=2017-09-26+09:59:45&from=%2B43676-----&to=%2B43668-----&charset=UTF-8& ←
  coding=0&text=...':
2017-09-26 11:59:45 [22763] [10] DEBUG:   Scheme: https://
2017-09-26 11:59:45 [22763] [10] DEBUG:   Host: 192.168.1.2
2017-09-26 11:59:45 [22763] [10] DEBUG:   Port: 1443
2017-09-26 11:59:45 [22763] [10] DEBUG:   Username: (null)
2017-09-26 11:59:45 [22763] [10] DEBUG:   Password: (null)
2017-09-26 11:59:45 [22763] [10] DEBUG:   Path: /internalsms/receive
2017-09-26 11:59:45 [22763] [10] DEBUG:   Query: auth_token=fNLosMgwdNUrKvEfmM9& ←
  timestamp=2017-09-26+09:59:45&from=%2B43676-----
&to=%2B43668-----&charset=UTF-8&coding=0&text=...
2017-09-26 11:59:45 [22763] [10] DEBUG:   Fragment: (null)
2017-09-26 11:59:45 [22763] [10] DEBUG: Connecting nonblocking to <192.168.1.2>
2017-09-26 11:59:45 [22763] [10] DEBUG: HTTP: Opening connection to '192.168.1.2:1443' ( ←
  fd=31).
2017-09-26 11:59:45 [22763] [10] DEBUG: Socket connecting
2017-09-26 11:59:45 [22763] [9]  DEBUG: Get info about connecting socket
2017-09-26 11:59:45 [22763] [9]  DEBUG: HTTP: Sending request:
2017-09-26 11:59:45 [22763] [9]  DEBUG: Octet string at 0x7f5dbc00f470:
2017-09-26 11:59:45 [22763] [9]  DEBUG:   len:   382
2017-09-26 11:59:45 [22763] [9]  DEBUG:   size: 1024
2017-09-26 11:59:45 [22763] [9]  DEBUG:   immutable: 0
2017-09-26 11:59:45 [22763] [9]  DEBUG:   data: 47 45 54 20 2f 69 6e 74 65 72 6e 61 6c 73 ←
  6d 73   GET /internalsms
2017-09-26 11:59:45 [22763] [9]  DEBUG:   data: 2f 72 65 63 65 69 76 65 3f 61 75 74 68 5f ←
  74 6f   /receive?auth_to
2017-09-26 11:59:45 [22763] [9]  DEBUG:   data: 6b 65 6e 3d ... ←
  ken=
  ... 20 48 54 54 50 2f 31 2e 31 ←
  0d 0a           HTTP/1.1..
2017-09-26 11:59:45 [22763] [9]  DEBUG:   data: 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 ←
  65 70   Connection: keep
2017-09-26 11:59:45 [22763] [9]  DEBUG:   data: 2d 61 6c 69 76 65 0d 0a 55 73 65 72 2d 41 ←
  67 65   -alive..User-Age
2017-09-26 11:59:45 [22763] [9]  DEBUG:   data: 6e 74 3a 20 4b 61 6e 6e 65 6c 2f 31 2e 34 ←
  2e 34   nt: Kannel/1.4.4
2017-09-26 11:59:45 [22763] [9]  DEBUG:   data: 0d 0a 48 6f 73 74 3a 20 31 39 32 2e 31 36 ←
  38 2e   ..Host: 192.168.
2017-09-26 11:59:45 [22763] [9]  DEBUG:   data: 31 2e 32 3a 31 34 34 33 0d 0a 0d 0a ←
  1.2:1443....
2017-09-26 11:59:45 [22763] [9]  DEBUG: Octet string dump ends.
2017-09-26 11:59:45 [22763] [9]  DEBUG: HTTP: Status line: <HTTP/1.1 200 OK>
2017-09-26 11:59:45 [22763] [9]  DEBUG: HTTP: Received response:
2017-09-26 11:59:45 [22763] [9]  DEBUG: Octet string at 0x7f5dbc006970:
2017-09-26 11:59:45 [22763] [9]  DEBUG:   len:   333
2017-09-26 11:59:45 [22763] [9]  DEBUG:   size: 1024
2017-09-26 11:59:45 [22763] [9]  DEBUG:   immutable: 0
2017-09-26 11:59:45 [22763] [9]  DEBUG:   data: 53 65 72 76 65 72 3a 20 6e 67 69 6e 78 0d ←

```

```

0a 44  Server: nginx..D
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 61 74 65 3a 20 54 75 65 2c 20 32 36 20 53 ←
65 70  ate: Tue, 26 Sep
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 20 32 30 31 37 20 30 39 3a 35 39 3a 34 35 ←
20 47  2017 09:59:45 G
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 4d 54 0d 0a 43 6f 6e 74 65 6e 74 2d 54 79 ←
70 65  MT..Content-Type
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 3a 20 74 65 78 74 2f 68 74 6d 6c 3b 20 63 ←
68 61  : text/html; cha
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 72 73 65 74 3d 75 74 66 2d 38 0d 0a 43 6f ←
6e 74  rset=utf-8..Cont
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 65 6e 74 2d 4c 65 6e 67 74 68 3a 20 30 0d ←
0a 43  ent-Length: 0..C
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 65 ←
70 2d  onnection: keep-
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 61 6c 69 76 65 0d 0a 53 65 74 2d 43 6f 6f ←
6b 69  alive..Set-Cooki
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 65 3a 20 6e 67 63 70 5f 70 61 6e 65 6c 5f ←
73 65  e: ngcp_panel_se
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 73 73 69 6f 6e 3d 34 35 30 32 64 64 66 65 ←
31 62  ssion=4502ddfelb
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 63 31 65 33 39 30 65 30 64 36 66 39 64 34 ←
37 30  c1e390e0d6f9d470
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 35 30 37 62 64 64 33 61 65 32 36 62 64 63 ←
3b 20  507bdd3ae26bdc;
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 70 61 74 68 3d 2f 3b 20 65 78 70 69 72 65 ←
73 3d  path=/; expires=
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 54 75 65 2c 20 32 36 2d 53 65 70 2d 32 30 ←
31 37  Tue, 26-Sep-2017
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 20 31 30 3a 35 39 3a 34 35 20 47 4d 54 3b ←
20 48  10:59:45 GMT; H
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 74 74 70 4f 6e 6c 79 0d 0a 58 2d 43 61 74 ←
61 6c  ttpOnly..X-Catal
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 79 73 74 3a 20 35 2e 39 30 30 37 35 0d 0a ←
53 74  yst: 5.90075..St
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 72 69 63 74 2d 54 72 61 6e 73 70 6f 72 74 ←
2d 53  rict-Transport-S
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 65 63 75 72 69 74 79 3a 20 6d 61 78 2d 61 ←
67 65  ecurity: max-age
2017-09-26 11:59:45 [22763] [9] DEBUG:  data: 3d 31 35 37 36 38 30 30 30 0d 0a 0d 0a ←
=15768000....
2017-09-26 11:59:45 [22763] [9] DEBUG: Octet string dump ends.
2017-09-26 11:59:45 [22763] [6] WARNING: Tried to set Coding field, denied.
2017-09-26 11:59:45 [22763] [6] INFO: No reply sent, denied.
2017-09-26 11:59:55 [22763] [9] DEBUG: HTTP: Server closed connection, destroying it ←
<192.168.1.2:1443:1::><0x7f5db0000b20><fd:31>.
...

```

- **short log of sent/received messages:** /var/log/kannel/smsbox-access.log

```
...
2017-09-26 12:39:18 SMS HTTP-request sender:+43680----- request: '' url: 'https ↵
://192.168.1.2:1443/internalsms/receive?
auth_token=fNLosMgwdNURkVefFMm9&timestamp=2017-09-26+10:39:18&from=%2B43680-----&to=%2 ↵
B43668-----&charset=UTF-8&coding=0
&text=<...message content...>' reply: 200 '<< successful >>'
...
2017-09-26 12:41:54 send-SMS request added - sender:sipwise:43668----- 192.168.1.3 ↵
target:43680----- request: '<...message content...>'
...
```

### 5.27.3 REST API

Handling of short messages from the user perspective happens with the help of NGCP's REST API. There is a dedicated resource: <https://<IP of WEB node>:1443/api/sms> that allows you to:

- **Get a list of sent and received messages.** This is achieved by sending a GET request on the /api/sms collection, as in the following example:

```
curl -i -X GET -H 'Connection: close' --cert NGCP-API-client-certificate.pem \
--cacert ca-cert.pem 'https://example.org:1443/api/sms/?page=1&rows=10'
```

- **Retrieve an SM** (both sent and received). This is achieved by sending a GET request for a specific /api/sms/id item, as in the following example:

```
curl -i -X GET -H 'Connection: close' --cert NGCP-API-client-certificate.pem \
--cacert ca-cert.pem 'https://example.org:1443/api/sms/1'
```

- **Send a new message** from a local subscriber. This is achieved by sending a POST request for the /api/sms collection, as in the following example:

```
curl -i -X POST -H 'Connection: close' -H 'Content-Type: application/json' \
--cert NGCP-API-client-certificate.pem --cacert ca-cert.pem \
'https://example.org:1443/api/sms/' --data-binary '{"callee" : "43555666777", \
"subscriber_id" : 4, "text" : "test"}'
```

As always, the full documentation of the REST API resources is available on the admin web interface of NGCP: <https://<IP of WEB node>:1443/api/#sms>



## 6 Customer Self-Care Interface and Menus

There are two ways for end users to maintain their subscriber settings: via the *Customer Self-Care Web Interface* and via *Vertical Service Codes* using their SIP phones.

### 6.1 The Customer Self-Care Web Interface

The NGCP provides a web panel for end users (CSC panel) to maintain their subscriber accounts, which is running on *https://<ngcp-ip>*. Every subscriber can log in there, change subscriber feature settings, view their call lists, retrieve voicemail messages and trigger calls using the click-to-dial feature.

#### 6.1.1 Login Procedure

To log into the CSC panel, the end user has to provide his full web username (e.g. *user1@1.2.3.4*) and the web password defined in Section 4.3. Once logged in, he can change his web password in the *Account* section. This will NOT change his SIP password, so if you control the end user devices, you can auto-provision the SIP password into the device and keep it secret, and just hand over the web password to the customer. This way, the end user will only be able to place calls with this auto-provisioned device and not with an arbitrary soft-phone, but can nonetheless manage his account via the CSC panel.

#### 6.1.2 Site Customization

As an operator (as well as a Reseller), you can change the branding logo of the Customer Self-Care (CSC) panel and the available languages on the CSC panel. This is possible via the admin web interface.

##### 6.1.2.1 Changing the Logo

For changing the branding logo on a reseller's admin web page and on the CSC panel you just need to access the web interface **as Administrator** and navigate to *Reseller* menu. Once there click on the *Details* button for your selected reseller, finally select *Branding*.

In order to do the same **as Reseller**, login on the admin web interface with the reseller's web credentials, then access the *Panel Branding* menu.

The web panel customisation happens as follows:

1. Press the *Edit Branding* button to start the customisation process.
2. Press the *Browse* button to select an image for the new logo:

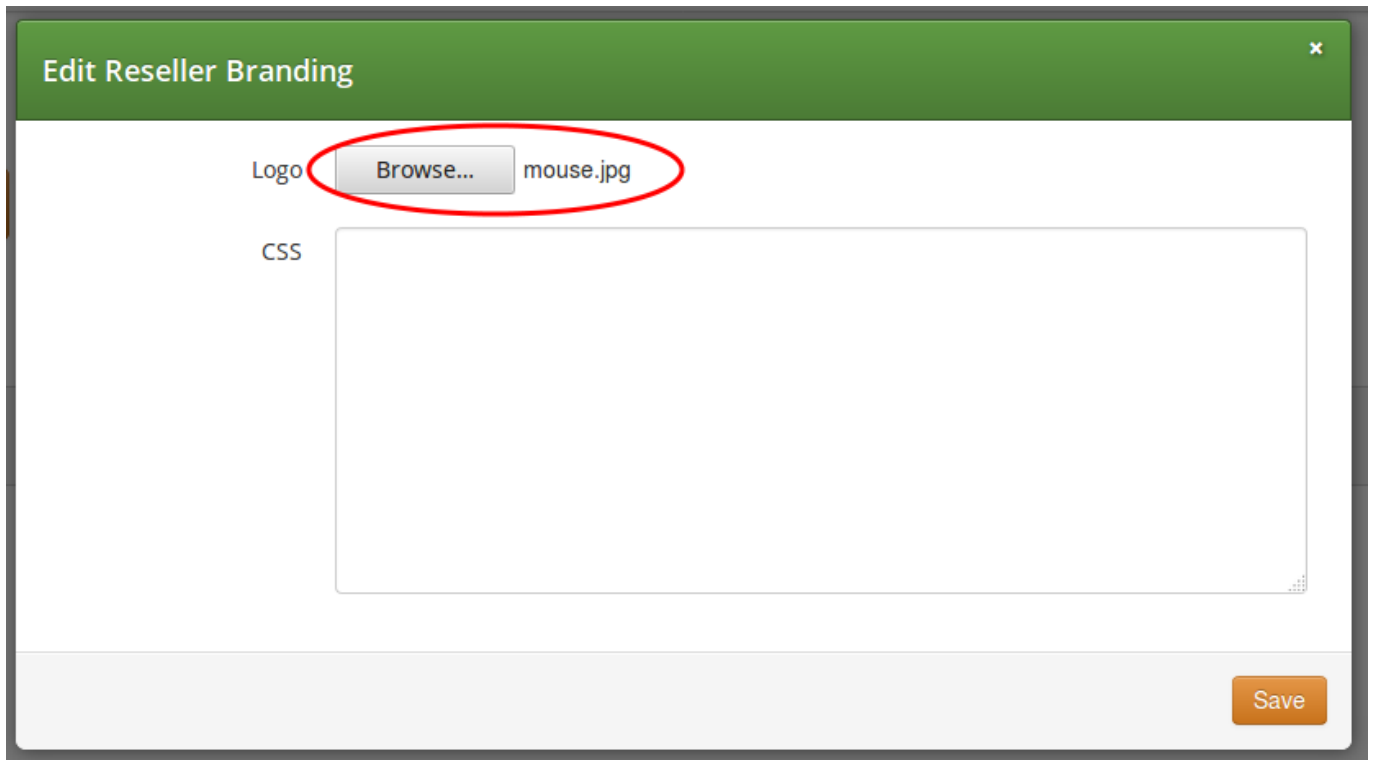



Figure 64: CSC Customisation Step 1: Select an image

3. Press the *Save* button to save changes.
4. Select and copy the auto-generated CSS code from the text box below the uploaded image:

Reseller branding successfully updated

[Edit Branding](#) [Delete Logo](#)

### Custom Logo



You can use the logo by adding the following CSS to the Custom CSS below.

```
#header .brand {  
  background: url(https://10.15.18.227:1443/reseller/3/css/logo/download) no-repeat 0 0;  
  background-size: 280px 32px;  
}
```

### Custom CSS

Figure 65: CSC Customisation Step 2: Copy CSS code

5. Press the *Edit Branding* button again.
6. Paste the CSS code into CSS text box and Save the changes:

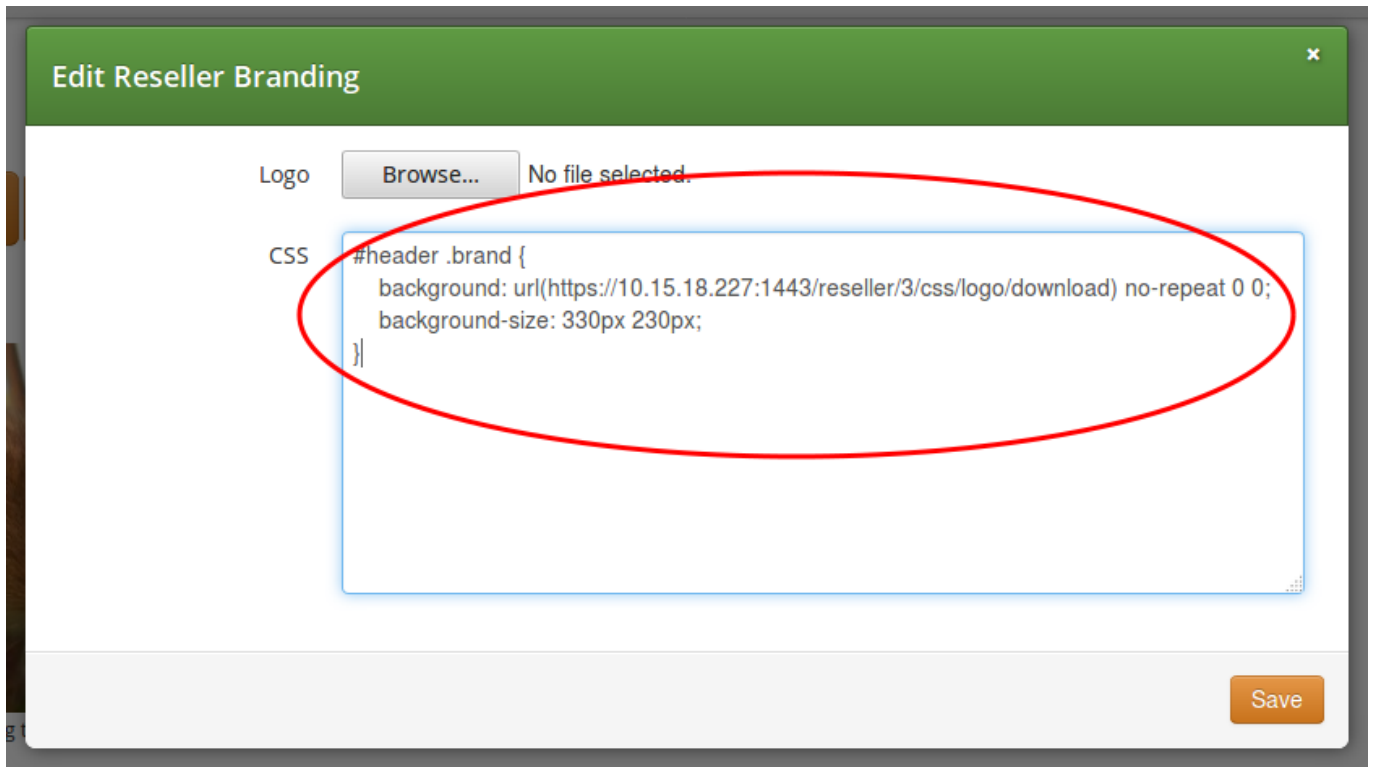


Figure 66: CSC Customisation Step 3: Paste CSS code

7. Now the new logo is already visible on the admin / CSC panel. If you want to hide the Sipwise copyright notice at the bottom of the web panels, add a line of CSS code as shown here:

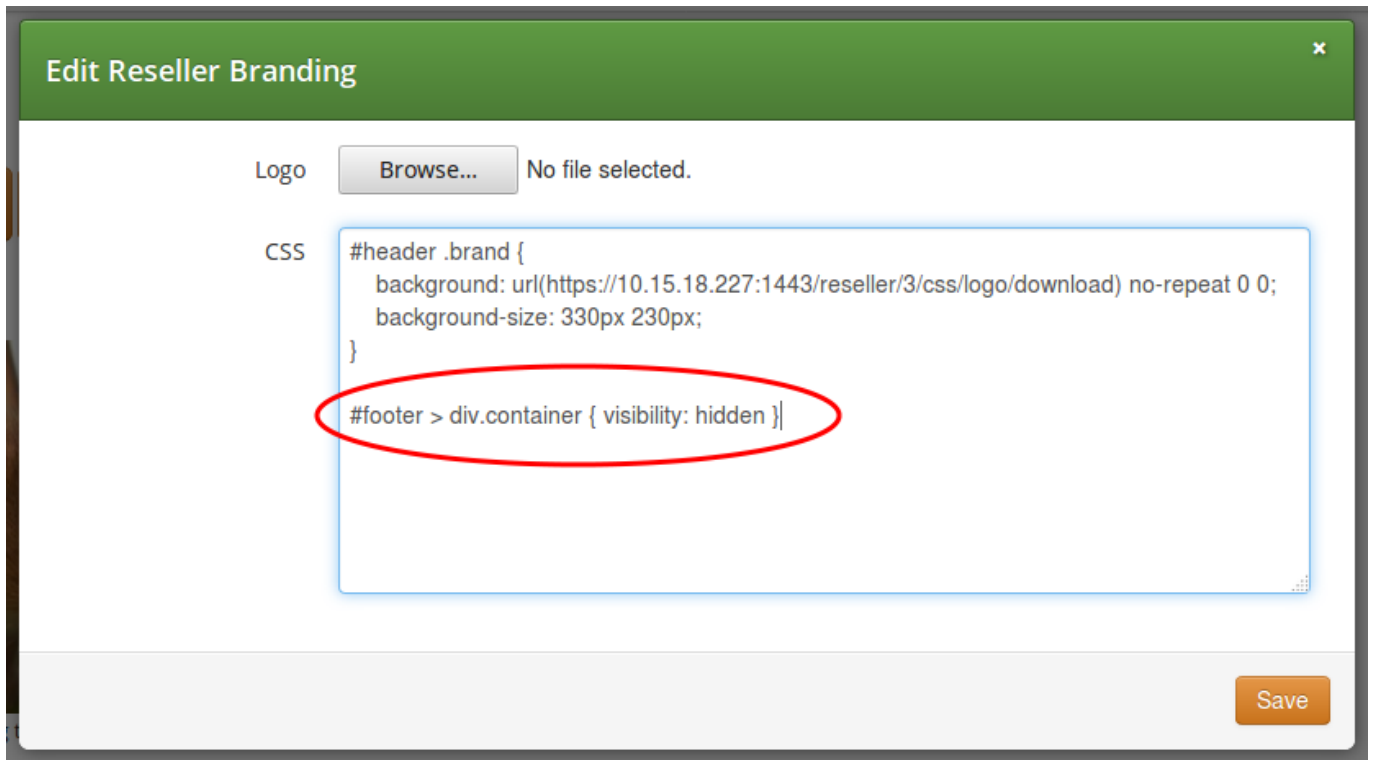


Figure 67: CSC Customisation: Hide copyright notice

8. The final branding data is shown on the admin web panel:

Reseller branding successfully updated

[Edit Branding](#) [Delete Logo](#)

Custom Logo



You can use the logo by adding the following CSS to the Custom CSS below:

```
#header .brand {
  background: url(https://10.15.18.227:1443/reseller/3/css/logo/download) no-repeat 0 0;
  background-size: 280px 32px;
}
```

Custom CSS

```
#header .brand {
  background: url(https://10.15.18.227:1443/reseller/3/css/logo/download) no-repeat 0 0;
  background-size: 330px 230px;
}

#footer > div.container { visibility: hidden }
```

Figure 68: CSC Customisation: Custom data on panel

### 6.1.2.2 Other Website Customisations

The layout and style of NGCP's admin and CSC web panel is determined by a single CSS file: `/usr/share/ngcp-panel/static/css/application.css`

More complex changes, like replacing colour of some web panel components, is possible via the modification of the CSS file.



#### Warning

Only experienced users with profound CSS knowledge are advised to change web panel properties in the main CSS file. *Sipwise does not recommend and also does not support the modification of the main CSS file.*

### 6.1.2.3 Selecting Available Languages

You can also enable/disable specific languages a user can choose from in the CSC panel. Currently, English (`en`), German (`de`), Italian (`it`), Spanish (`es`) and Russian (`ru`) are supported, and the default language is the same as the browser's preferred one.

You can select the *default language* provided by CSC by changing the parameter `www_admin.force_language` in `/etc/ngcp-config/config.yml` file. An example to set the English language as default: `force_language: en`

## 6.2 The Voicemail Menu

NGCP offers several ways to access the Voicemail box.

The CSC panel allows your users to listen to voicemail messages from the web browser, delete them and call back the user who left the voice message. User can setup voicemail forwarding to the external email and the PIN code needed to access the voicebox from any telephone also from the CSC panel.

**To manage the voice messages from SIP phone:** simply dial internal voicemail access number 2000.

To change the access number: look for the parameter *voicemail\_number* in */etc/ngcp-config/config.yml* in the section *sems*→*vsc*. After the changes, execute *ngcpcfg apply 'changed voicebox number'*.

---

### Tip

To let the callers leave a voice message when user is not available he should enable Call Forward to Voicebox. The Call Forward can be provisioned from the CSC panel as well as by dialing Call Forward VSC with the voicemail number. E.g. when parameter *voicemail\_number* is set to 9999, a Call Forward on Not Available to the Voicebox is set if the user dials \*93\*9999. As a result, all calls will be redirected to the Voicebox if SIP phone is not registered.

---

**To manage the voice messages from any phone:**

- As an operator, you can setup some DID number as external voicemail access number: for that, you should add a special rewrite rule (Inbound Rewrite Rule for Callee, see Section 4.7.) on the incoming peer, to rewrite that DID to "voiceboxpass". Now when user calls this number the call will be forwarded to the voicemail server and he will be prompted for mailbox and password. The mailbox is the full E.164 number of the subscriber account and the password is the PIN set in the CSC panel.
- The user can also dial his own number from PSTN, if he setup Call Forward on Not Available to the Voicebox, and when reaching the voicemail server he can interrupt the "user is unavailable" message by pressing \* key and then be prompted for the PIN. After entering PIN and confirming with # key he will enter own voicemail menu. PIN is random by default and must be kept secret for that reason.

## 7 Billing Configuration

This chapter describes the steps necessary to rate calls and export rated CDRs (call detail records) to external systems.

### 7.1 Billing Profiles

Service billing on the NGCP is based on billing profiles, which may be assigned to customers and SIP peerings. The design focuses on a simple, yet flexible approach, to support arbitrary dial-plans without introducing administrative overhead for the system administrators. The billing profiles may define a base fee and free time or free money per billing interval. Unused free time or money automatically expires at the end of the billing interval.

Each profile may have call destinations (usually based on E.164 number prefix matching) with configurable fees attached. Call destination fees each support individual intervals and rates, with a different duration and/or rate for the first interval. (e.g.: charge the first minute when the call is opened, then every 30 seconds, or make it independent of the duration at all) It is also possible to specify different durations and/or rates for peak and off-peak hours. Peak time may be specified based on weekdays, with additional support for manually managed dates based on calendar days. The call destinations can finally be grouped for an overview on user's invoices by specifying a zone in two detail levels. (E.g.: national landline, national mobile, foreign 1, foreign 2, etc.)

#### 7.1.1 Creating Billing Profiles

The first step when setting up billing data is to create a billing profile, which will be the container for all other billing related data. Go to *Settings*→*Billing* and click on *Create Billing Profile*.



Logged in as administrator Logout

### Create Billing Profiles

Reseller Search:

#	Name	Contract #	Status	
1	default	1	active	1 <input checked="" type="checkbox"/>

Showing 1 to 1 of 1 entries

Create Reseller

Handle  2

Name  3

Prepaid

Interval charge

4

The fields *Reseller*, *Handle* and *Name* are mandatory.

- **Reseller:** The reseller this billing profile belongs to.
- **Handle:** A unique, permanently fixed string which is used to attach the billing profile to a customer or SIP peering contract.
- **Name:** A free form string used to identify the billing profile in the *Admin Panel*. This may be changed at any time.
- **Interval charge:** A base fee for the billing interval, specifying a monetary amount (represented as a floating point number) in whatever currency you want to use.
- **Interval free time:** If you want to include free calling time in your billing profile, you may specify the number of seconds that are available every billing interval. See *Creating Billing Fees* below on how to select destinations which may be called using the free time.
- **Interval free cash:** Same as for *interval free time* above, but specifies a monetary amount which may be spent on outgoing calls. This may be used for example to implement a minimum turnover for a contract, by setting the *interval charge* and *interval free cash* to the same values.
- **Fraud monthly limit:** The monthly fraud detection limit (in Cent) for accounts with this billing profile. If the call fees of an account reach this limit within a billing interval, an action can be triggered.
- **Fraud monthly lock:** a choice of *none*, *foreign*, *outgoing*, *incoming*, *global*. Specifies a lock level which will be used to lock the account and his subscribers when *fraud monthly limit* is exceeded.
- **Fraud monthly notify:** An email address or comma-separated list of email addresses that will receive notifications when *fraud monthly limit* is exceeded.

- **Fraud daily limit:** The fraud detection limit (in Cent) for accounts with this billing profile. If the call fees of an account reach this limit within a calendar day, an action can be triggered.
- **Fraud daily lock:** a choice of *none*, *foreign*, *outgoing*, *incoming*, *global*. Specifies a lock level which will be used to lock the account and his subscribers when *fraud daily limit* is exceeded.
- **Fraud daily notify:** An email address or comma-separated list of email addresses that will receive notifications when *fraud daily limit* is exceeded.
- **Currency:** The currency symbol for your currency. Any UTF-8 character may be used and will be printed in web interfaces.
- **VAT rate:** The percentage of value added tax for all fees in the billing profile. Currently for informational purpose only and not used further.
- **VAT included:** Whether VAT is included in the fees entered in web forms or uploaded to the platform. Currently for informational purpose only and not used further.

### 7.1.2 Creating Billing Fees

Each *Billing Profile* holds multiple *Billing Fees*.

To set up billing fees, click on the *Fees* button of the billing profile you want to configure. Billing fees may be uploaded using a configurable CSV file format, or entered directly via the web interface by clicking *Create Fee Entry*. To configure the CSV field order for the file upload, rearrange the entries in the `www_admin→fees_csv→element_order` array in `/etc/ngcp-config/config.yml` and execute the command `ngcpcfg apply changed fees element order`. The following is an example of working CSV file to upload (pay attention to double quotes):

```
".", "^1", out, "EU", "ZONE EU", 5.37, 60, 5.37, 60, 5.37, 60, 5.37, 60, 0, 0
"^01.+$", "^02145.+$", out, "AT", "ZONE Test", 0.06250, 1, 0.06250, 1, 0.01755, 1, 0.01733, 1, 0
```

For input via the web interface, just fill in the text fields accordingly.

Zone

Search:

#	Zone	Zone Detail	
2	test	test zone	2 <input checked="" type="checkbox"/>

Showing 1 to 1 of 1 entries

created by "Create Zone" button below

1

Source

3 Destination

4 Direction

Onpeak init rate

In both cases, the following information may be specified independently for every destination:

- **Zone:** A zone for a group of destinations. May be used to group destinations for simplified display, e.g. on invoices. (e.g. foreign zone 1)
- **Source:** The source pattern. This is a POSIX regular expression matching the complete source URI (e.g. `^.*@sip\.example\.org$` or `^someone@sip\.sipwise\.com$` or just `.` to match everything). If you leave this field empty, the default pattern `.` matching everything will be set implicitly. Internally, this pattern will be matched against the `<source_cli>`@`<source_domain>` fields of the CDR.
- **Destination:** The destination pattern. This is a POSIX regular expression matching the complete destination URI (e.g. `someone@sip\.example\.org` or `^43`). This field must be set.
- **Direction:** `Outbound` for standard origination fees (applies to callers placing a call and getting billed for that) or `Inbound` for termination fees (applies to callees if you want to charge them for receiving various calls, e.g. for 800-numbers). *If in doubt, use Outbound.* If you upload fees via CSV files, use `out` or `in`, respectively.



#### Important

The {source, destination, direction} combination needs to be unique for a billing profile. The system will return an error if such a set is specified twice, both for the file upload and the input via the web interface.

**Important**

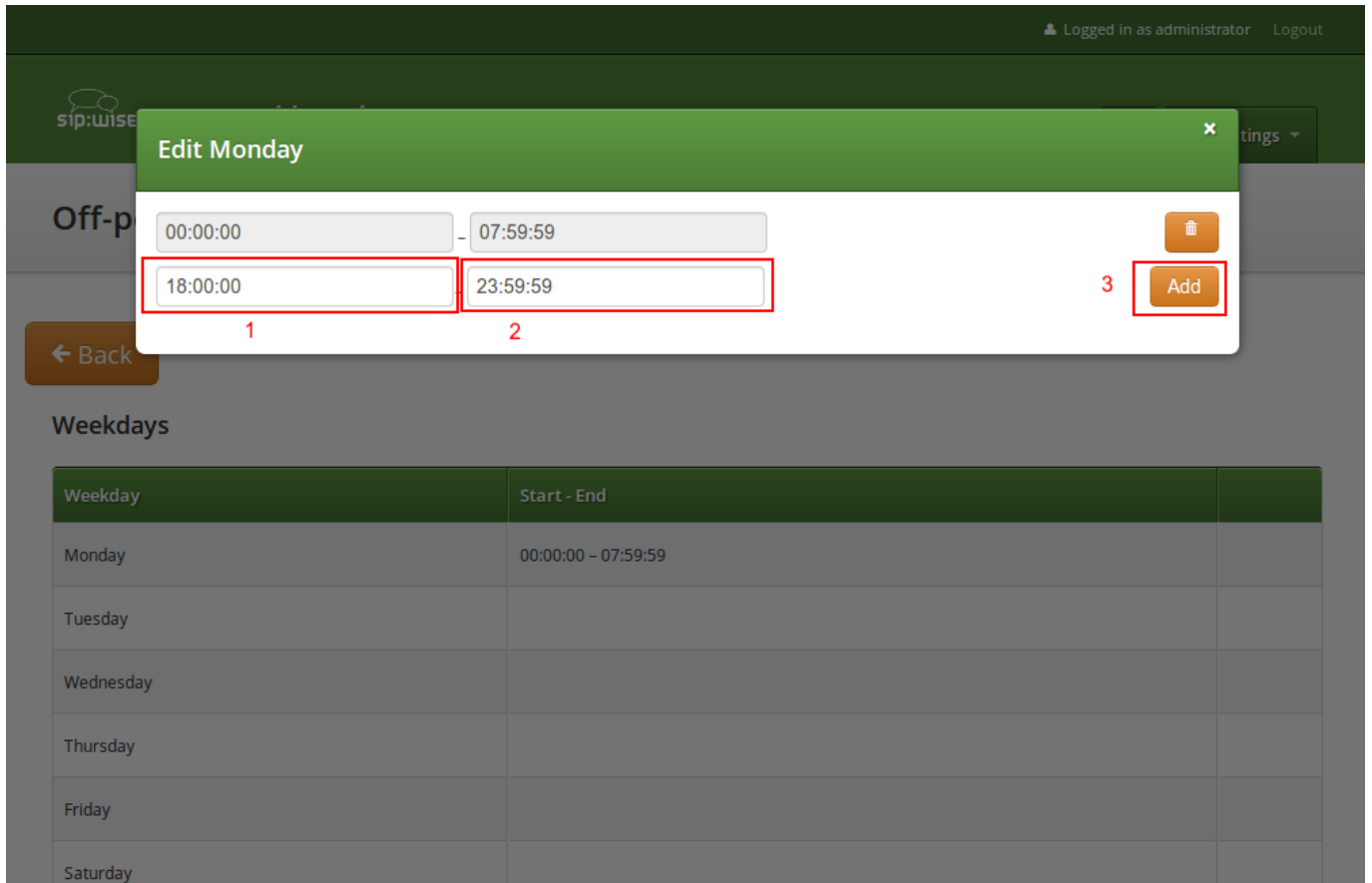
There are several internal services (vsc, conference, voicebox) which will need a specific destination entry with a domain-based destination. If you don't want to charge the same (or nothing) for those services, add a fee for destination `\.local$` there. If you want to charge different amounts for those services, break it down into separate fee entries for `@vsc\.local$`, `@conference\.local$` and `@voicebox\.local$` with the according fees. **NOT CREATING EITHER THE CATCH-ALL FEE OR THE SEPARATE FEES FOR THE `.local` DOMAIN WILL BREAK YOUR RATING PROCESS!**

- **Onpeak init rate:** The rate for the first rating interval in cent (of whatever currency, represented as a floating point number) per second. Applicable to calls during onpeak hours.
- **Onpeak init interval:** The duration of the first billing interval, in seconds. Applicable to calls during onpeak hours.
- **Onpeak follow rate:** The rate for subsequent rating intervals in cent (of whatever currency, represented as a floating point number) per second. Applicable to calls during onpeak hours. Defaults to *onpeak init rate*.
- **Onpeak follow interval:** The duration of subsequent billing intervals, in seconds. Applicable to calls during onpeak hours. Defaults to *onpeak init interval*.
- **Offpeak init rate:** The rate for the first rating interval in cent (of whatever currency, represented as a floating point number) per second. Applicable to calls during off-peak hours. Defaults to *onpeak init rate*.
- **Offpeak init interval:** The duration of the first billing interval, in seconds. Applicable to calls during off-peak hours. Defaults to *onpeak init interval*.
- **Offpeak follow rate:** The rate for subsequent rating intervals in cent (of whatever currency, represented as a floating point number) per second. Applicable to calls during off-peak hours. Defaults to *offpeak init rate* if that one is specified, or to *onpeak follow rate* otherwise.
- **Offpeak follow interval:** The duration of subsequent billing intervals, in seconds. Applicable to calls during off-peak hours. Defaults to *offpeak init interval* if that one is specified, or to *onpeak follow interval* otherwise.
- **Use free time:** Specifies whether free time minutes may be used when calling this destination. May be specified in the file upload as 0, n[o], f[alse] and 1, y[es], t[rue] respectively.

**7.1.3 Creating Off-Peak Times**

To be able to differentiate between on-peak and off-peak calls, the platform stores off-peak times for every billing profile based on weekdays and/or calendar days. To edit the settings for a billing profile, go to *Settings*→*Billing* and press the *Off-Peaktimes* button on the billing profile you want to configure.

To set off-peak times for a weekday, click on *Edit* next to the according weekday. You will be presented with two input fields which both receive a timestamp in the form of *hh:mm:ss* specifying a time of day for the start and end of the off-peak period. If any of the fields is left empty, the system will automatically insert *00:00:00* (*start* field) or *23:59:59* (*end* field). Click on *Add* to store the setting in the database. You may create more than one off-peak period per weekday. To delete a range, just click *Delete* next to the entry. Click the *close* icon when done.



To specify off-peak ranges based on calendar dates, click on *Create Special Off-Peak Date*. Enter a date in the form of *YYYY-MM-DD hh:mm:ss* into the *Start Date/Time* input field and *End Date/Time* input field to define a range for the off-peak period.

1 Start Date/Time 2013-12-24 00:00:00

2 End Date/Time 2013-12-24 23:59:59

3 Save

Weekday	Start - End
Monday	00:00:00 - 07:59:59 18:00:00 - 23:59:59
Tuesday	
Wednesday	
Thursday	
Friday	

## 7.2 Peak Time Call Rating Modes

### 7.2.1 Introduction to Call Rating Modes

The call rating engine component (rate-o-mat) supports two different modes to consider configured off-peak/on-peak periods when calculating call costs:

- **Split-Peak-Parts mode:** CDRs reflecting calls which cross an off-peak/on-peak period transition will be split into two CDR fragments. This way it is possible for each fragment to exactly mark it as either on-peak or off-peak, and the CDR's *frag\_carrier\_onpeak*, *frag\_reseller\_onpeak* and *frag\_customer\_onpeak* fields can be populated accordingly.

CDRs that are entirely within either on-peak or off-peak periods are not split and show a value of 0 for their *is\_fragmented* field. CDR fragments are marked by the *is\_fragmented* field showing a value of 1. If the call is crossing  $n$  transitions,  $(n+1)$  fragments are created.

Apart from *is\_fragmented*, *\*\_onpeak* and *\*\_cost* fields, each fragment is a copy of the original CDR, except for *start\_time* and *duration* fields. The sum of *duration* of fragments is equal to the *duration* of the original CDR. Fragments are adjacent, so the *start\_time* of a fragment is equal to the end time (*start\_time* + *duration*) of the previous fragment.

- **Regular mode:** In regular mode, the costs are calculated by summing up init/follow interval ticks, and selecting on-peak or off-peak rates of the billing fee per tick. Resulting call costs will be identical to the sum of the costs of fragmented CDRs in Split-Peak-Parts mode, but now comprised of both on-peak and off-peak rates in a single value. Hence *frag\_carrier\_onpeak*, *frag\_reseller\_onpeak* and *frag\_customer\_onpeak* CDR fields cannot be provided.

## 7.2.2 Typical Use Cases for Call Rating Modes

The CDR fragmentation produced by **Split-Peak-Parts mode** can be useful when implementing:

- End-customer invoicing to **separate** call listings or costs by **off-peak and on-peak**
- Reports to compare sums of carrier and customer costs when fees with **different metering** (given by the fees' init and follow interval) are in effect

The process of the **regular mode** does not create additional CDRs, which has advantages in other situations:

- It is easy to **re-rate** CDRs, as there is no need to revert fragmentation.
- The concept of **one-CDR-per-call-leg** is kept, which simplifies external rating, reporting, call-flow visualisation etc.

## 7.2.3 Configuration of Call Rating Modes

The regular mode is enabled by default. To enable Split-Peak-Parts mode, set `rateomat.splitpeakparts` to 1 in `/etc/ngcp-config/config.yml` file.

## 7.3 Prepaid Accounting

In a normal post-paid accounting scenario, each customer accumulates debt in their billing account, which at the end of the billing interval is then billed to the customer. A *prepaid* billing profile reverses this sequence: the customer first has to provide credit to their account balance, and the costs for all calls are then deducted from that account balance. Once the balance reaches zero, no further calls from this customer are accepted, with the exception of free calls. Additionally, if the balance drops to zero while any calls are currently active, NGCP will disconnect those calls as soon as that happens.

With prepaid billing enabled, all details of the billing profile and all details of the billing fees behave as they normally do, including interval free time. If any interval free time is given, the free time will be used before the account's credit is.

---

### Important



For technical reasons, the system can make the distinction between on-peak and off-peak times only at call establishment time. In other words, if the currently active call fee at the moment when the call is established is an off-peak fee, then the same off-peak fee will remain active for the whole length of this call, even if the call actually transitions into an on-peak fee (and vice versa).



### Important

For technical reasons, prepaid billing can't charge local endpoint calls to Voicebox, VSC calls or calls to a Conference Room.

---

The Sipwise NGCP platform offers advanced billing features which are especially designed for pre-paid billing scenarios. For details please visit [Billing Customizations](#) Section 7.5 section of the handbook.

## 7.4 Fraud Detection and Locking

The NGCP supports a fraud detection feature, which is designed to detect accounts causing unusually high customer costs, and then to perform one of several actions upon those accounts. This feature can be enabled and configured through two sets of billing profile options described in Section 7.1.1, namely the monthly (*fraud monthly limit*, *fraud monthly lock* and *fraud monthly notify*) and daily limits (*fraud daily limit*, *fraud daily lock* and *fraud daily notify*). Either monthly/daily limits or both of them can be active at the same time.

Monthly fraud limit check runs once a day, shortly after midnight local time and daily fraud limit check runs every 30min. A background script (managed by cron daemon) automatically checks all accounts which are linked to a billing profile enabled for fraud detection, and selects those which have caused a higher cost than the *fraud monthly limit* configured in the billing profile, within the currently active billing interval (e.g. in the current month), or a higher cost than the *fraud daily limit* configured in the billing profile, within the calendar day. It then proceeds to perform at least one of the following actions on those accounts:

- If **fraud lock** is set to anything other than *none*, it will lock the account accordingly (e.g. if **fraud lock** is set to *outgoing*, the account will be locked for all outgoing calls).
- If anything is listed in **fraud notify**, an email will be sent to the email addresses configured. The email will contain information about which account is affected, which subscribers within that account are affected, the current account balance and the configured fraud limit, and also whether or not the account was locked in accordance with the **fraud lock** setting. It should be noted that this email is meant for the administrators or accountants etc., and not for the customer.

### 7.4.1 Fraud Lock Levels

Fraud lock levels are various protection (and notification) settings that are applied to subscribers of a *Customer*, if fraud detection is enabled in the currently active billing profile and the *Customer's* daily or monthly fraud limit has been exceeded.

The following lock levels are available:

- *none*: no account locking will happen
- *foreign calls*: only calls within the subscriber's own domain, and emergency calls, are allowed
- *all outgoing calls*: subscribers of the customer cannot place any calls, except calls to free and emergency destinations
- *incoming and outgoing*: subscribers of the customer cannot place and receive any calls, except calls to free and emergency destinations
- *global*: same restrictions as at *incoming and outgoing* level, additionally subscribers are not allowed to access the Customer Self Care (CSC) interface
- *ported*: only automatic call forwarding, due to number porting, is allowed



#### Important

You can override fraud detection and locking settings of a billing profile on a per-account basis via REST API or the Admin interface.

---



**Caution**

Accounts that were automatically locked by the fraud detection feature will **not** be automatically unlocked when the next billing interval starts. This has to be done manually through the administration panel or through the provisioning interface.

---

**Important**

If fraud detection is configured to only send an email and not lock the affected accounts, it will continue to do so for over-limit accounts every day. The accounts must either be locked in order to stop the emails (only currently active accounts are considered when the script looks for over-limit accounts) or some other action to resolve the conflict must be taken, such as disabling fraud detection for those accounts.

---

**Note**

It is possible to fetch the list of fraud events and thus get fraud status of *Customers* by using the REST API and referring to the resource: `/api/customerfraudevents`.

---

**Note**

Apart from the daily fraud detection check service, NGCP also provides instant, "hard" locking for prepaid use cases, by means of billing profile packages. See [Billing Profile Packages](#) Section 7.5.3 for reference.

---

## 7.5 Billing Customizations

The standard way of doing the billing—i.e. having fixed billing intervals of a calendar month, starting on the 1st day of month—may not fit all billing profiles and intervals that sip:carrier platform operators would like to use.

The sip:carrier supports—starting from its mr4.2.1 version—alternate ways of defining billing profiles and intervals which are especially worthy for pre-paid scenarios. New functionality is covered by the following titles:

1. [Billing Networks](#) Section 7.5.1
2. [Profile Mappings Schedule](#) Section 7.5.2
3. [Profile Packages](#) Section 7.5.3
4. [Vouchers](#) Section 7.5.4
5. [Top-up](#) Section 7.5.5
6. [Balance Overviews](#) Section 7.5.6
7. [Usage Examples](#) Section 7.5.7

Subsequent sections will provide an introduction and configuration instructions to these advanced features of sip:carrier.

---

### 7.5.1 Billing Networks

The idea is to dynamically select billing profiles (including fees) depending on the IP network the caller's SIP client is using to connect. The caller's IP is populated in a call's CDR, and effectively processed by:

- the rating engine component („rate-o-mat“) and the
- prepaid interception module (libswrate).

The billing profile for rating a call is identified by matching the source IP against network ranges linked to the customer contract's billing mappings records. This feature is sometimes also referred to as *roaming*.

A *Billing Network* is defined as a series of *network blocks* where each network block consists of a *single IP address* or an *IP subnet*. Blocks of a particular billing network can be defined by either IPv4, or IPv6 addresses but not mixed.

The screenshot shows a web application window titled "Create Billing Network". At the top, there is a "Reseller" section with a search bar and a table. The table has columns for "#", "Name", "Contract #", "Status", and an action icon. One row is visible with the following data: #16, Name "Demo Reseller", Contract # 200, Status "active". Below the table, it says "Showing 9 to 9 of 9 entries" and includes pagination controls. A "Create Reseller" button is located below the table. The main form area contains three input fields: "Billing Network Name" with the value "Demo Billing Net 1", "Description" with the value "Some text", and "Billing Network Block" with the value "10.0.1.0 / 24". There is a "Remove" button next to the "Billing Network Block" field. At the bottom right of the form, there is a "Save" button.

Figure 69: Creation of Billing Network

The new `/api/billingnetworks/` **REST API** resource allows to manage billing networks. The example billing network that is shown in the figure above may be defined through the API with this JSON structure:

```
{ "blocks" : [ { "ip" : "10.0.1.0", // subnet: 10.0.1.0 .. 10.0.1.255
                  "mask" : 24
```

```

    },
    { "ip" : "10.0.2.2" // single ip
    }
  ],
  "description" : "Some text",
  "name" : "Demo Billing Net 1", //unique per reseller
  "reseller_id" : 1
}

```

**Input validation** of the network blocks is automatically performed by sip:carrier during their definition in a way that it prevents specifying overlapping blocks by means of Interval Trees; billing networks themselves may overlap though.

The screenshot shows a web form titled "Create Billing Network". It contains the following fields and elements:

- Billing Network Name:** Demo Billing Net 3
- Description:** description net 3
- Billing Network Block 1:** 10.0.1.0 / 28 (with a "Remove" button)
- Billing Network Block 2:** 10.0.1.10 / 26 (with a "Remove" button)
- Error Message:** A red box at the bottom left states: "Block '10.0.1.10/26' overlaps with block(s) '10.0.1.0/28'".
- Action Buttons:** "Add another billing network block" and "Save" are located at the bottom right.

Figure 70: Overlapping Block Prevention

## 7.5.2 Profile Mapping Schedule

Using the default settings related to billing when creating a new *Reseller* or *Customer* on the administrative web panel results in applying the standard billing profile mapping schedule: the same billing profile is always used.

### 7.5.2.1 Definition of Profile Mapping Schedules

The idea of *billing profile mapping schedule* is to extend the billing mappings logic to utilize it as a schedule for billing profiles (and associated fees) for the *Customer* or *Reseller* contract. So far, billing mapping records provided only a history showing which profile was in effect at a given time in the past, which is for example required for delayed rating of calls.

Now it is also possible to define in advance, when specific billing profiles should become active in the future, e.g. to plan campaigns or special offers.

**Billing profile mappings** represent a schedule of overlapping time intervals with *Billing Profiles* and *Billing Networks*, which are assigned to (customer) contracts when creating or editing them.

*Mapping intervals* can be of type:

- open: no start time + no end time
- half-open:
  - left-open: no start time + definite end time
  - right-open: definite start time + no end time
- closed: definite start time + definite end time

### 7.5.2.2 Schedule Example

id	Billing Profile Interval Schedule Example	Mai 2015			Jun 2015										
		29	30	31	1	2	3	4	5	6	7	8	9	10	11
1	open: base/fallback (profile 1, no/any network)	[Active throughout the period]													
2	closed: (profile 2 , network 1) from June, 2nd. – 4th.	[Active from June 2nd to 4th]													
3	right open: (profile 3 , network 1) starting on June, 1st.	[Active from June 1st onwards]													
4	right open: (profile 4 , network 2) starting on June, 1st.	[Active from June 1st onwards]													
5	closed: (profile 5 , no/any network) from June, 3rd. – 10th.	[Active from June 3rd to 10th]													

Figure 71: Profile Mapping Schedule Example

Applying the profile mapping schedule shown in the above figure will result in billing profiles being active as provided in the table below.

Table 11: Active Billing Profiles

Table 11: (continued)

Time	Web Panel shows	Rating		
		Caller IP in Network 1	Caller IP in Network 2	Caller IP in other network
May 30	Profile 1	Profile 1	Profile 1	Profile 1
June 1	Profile 4	Profile 3	Profile 4	Profile 1
June 2	Profile 2	Profile 2	Profile 4	Profile 1
June 5	Profile 5	Profile 3	Profile 4	Profile 5

### 7.5.2.3 Configuration of Schedules

A Customer's default billing profile mapping can be changed to scheduled mappings when editing its properties, at the parameter "Set billing profiles", selecting: `schedule (billing mapping intervals)`

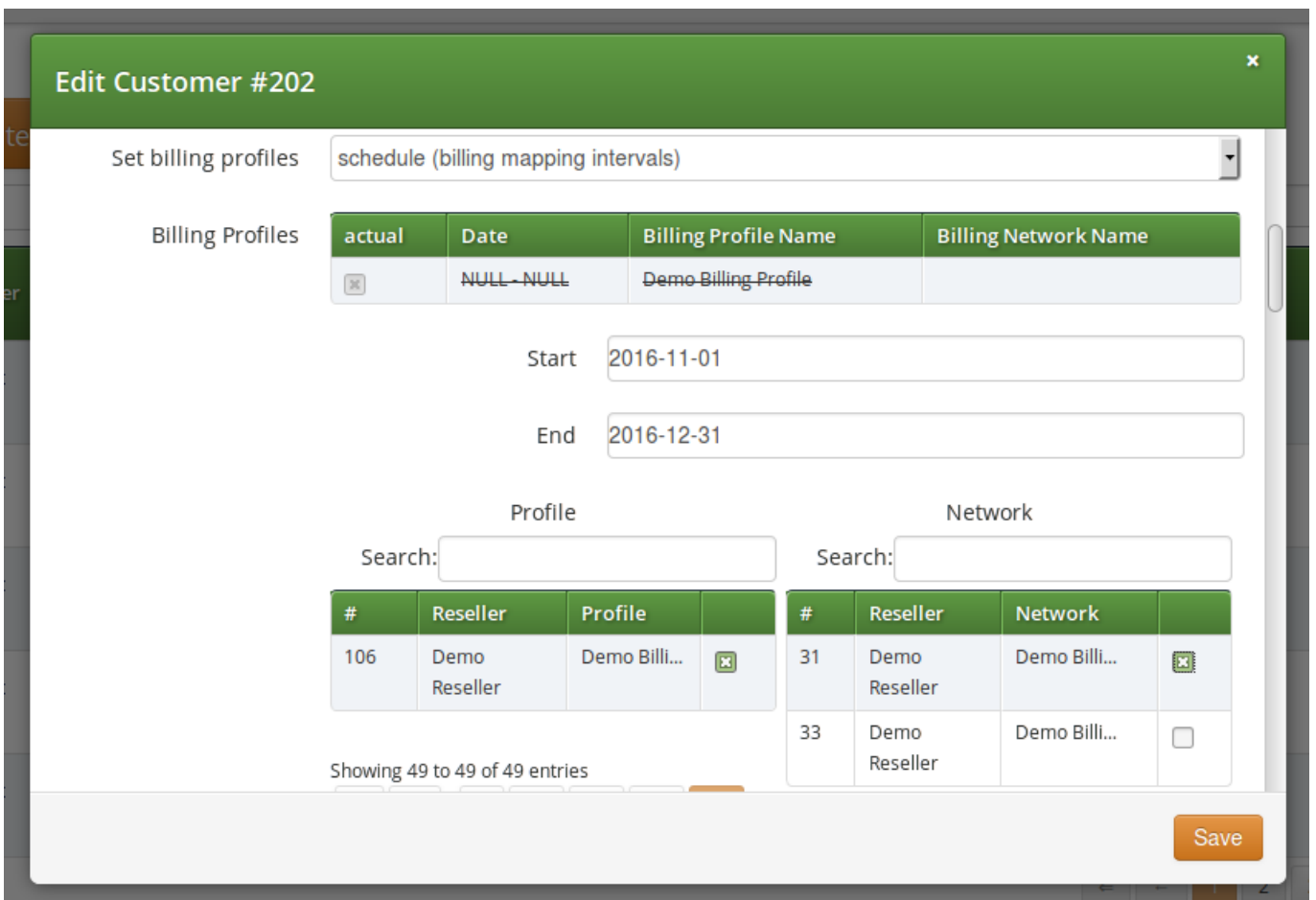


Figure 72: Profile Mapping Schedule Creation

**Tip**

Assigning a *Billing Network* to a billing profile mapping is optional. Without selecting the network, the *Billing Profile* will be applied to all calls.

The profile mapping schedule assigned to a *Customer* is also listed among *Customer's* properties. See *Settings* → *Customers* → *Details* → *Billing Profile Schedule*.

**Customer Details for #202 (Cloud PBX Account)**

← Back
☰ Preferences
✎ Edit

✦ Expand Groups

Reseller

Contact Details

Billing Profile Schedule

actual	Date	Billing Profile Name	Prepaid	Billing Network Name
<input checked="" type="checkbox"/>	NULL - NULL	Demo-Billing-Profile	<input type="checkbox"/>	
<input type="checkbox"/>	2016-11-01T00:00:00 - 2016-12-31T00:00:00	Demo Billing Profile	<input type="checkbox"/>	Demo Billing Net 1
<input type="checkbox"/>	2017-01-01T00:00:00 - 2017-12-31T00:00:00	Demo Billing Profile	<input type="checkbox"/>	

Subscribers

PBX Groups

Figure 73: Profile Mapping Schedule List

**Note**

Profile mappings that started in the past, like the default one, are displayed with a strike-through font in order to indicate that those can not be modified.

The currently active mapping is depicted by a checked box.

#### 7.5.2.4 REST API for Profile Mapping Schedules

The `/api/customers/` API resource was extended to provide three different modes of defining profile mappings:

1. `billing_profiles` field: explicitly declare profile mappings in form of (billing profile, billing network, start time, stop time) tuples
2. `billing_profile_id` field (legacy API spec): a single profile mapping interval is appended (billing profile, no network / any caller IP respectively, starting now)

3. `profile_package_id` field: profile mappings starting now are appended by using lists of (`billing profile`, `billing network`) tuples from the given profile package

With regards to *Resellers*, the `/api/contracts/` API resource was enhanced as well, but supports method 1. and 2. only, and without billing networks.

### Mapping Intervals

Intervals can be of open, half-open (left-open, right-open) or closed type. When specifying profile mappings discretely, allowed interval types are restricted, depending on create/update situation:

Table 12: Allowed Mapping Intervals

Interval Type	Start	Stop	POST (create)	PUT / PATCH (update)
open	undefined	undefined	1..*	0
left-open	undefined	defined	0	0
right-open	> now()	undefined	*	*
closed	> now()	> start	*	*

### Example Profile Mapping

An example JSON structure for definition of profile mapping schedules shown in [Billing Profile Schedule List Figure 73](#) :

```
{ ...,
  "billing_profile_definition" : "profiles", // i.e. use 'billing_profiles' field
  "billing_profiles" : [ { "network_id" : "236",
    "profile_id" : "236",
    "start" : "2016-11-01 00:00:00",
    "stop" : "2016-12-31 00:00:00"
  }, // closed future interval, with network
  { "network_id" : null,
    "profile_id" : "237",
    "start" : "2017-01-01 00:00:00",
    "stop" : "2017-12-31 00:00:00"
  } ], // closed future interval, without network
  "contact_id" : 141,
  ...
}
```

### 7.5.3 Profile Packages

By introducing billing profile packages, general billing parameters can be defined for a customer contract:

- Balance interval duration (regular/constant or aligned to top-up events)
- The first interval's start date
- The cash-balance carry-over/discard behaviour upon interval transitions
- Subscriber lock levels and profile sets to get applied upon:
  - top-up
  - balance threshold underrun
- Initial balance and billing profiles

*Profile Packages* are fundamental for pre-paid billing scenarios, since in such a billing scheme the traditional, fixed monthly periods prove to be insufficient to cover the business needs of the NGCP platform operator. As an example: pre-paid subscribers typically have their "billing periods" between account balance top-ups.

### 7.5.3.1 Elements of Profile Packages

A *Profile Package* consists of various elements that will be discussed in subsequent sections of the sip:carrier handbook. In order to set the parameters of a profile package one must navigate to: *Settings* → *Profile Packages* → *Create Profile Package*, or alternatively, in order to update an existing profile package: select the package and press *Edit* button.

#### **Basic Balance Intervals Setup**

- Interval duration (n hours, days, weeks, months)
- Interval start mode:
  - 1st of month (1st): billing interval is 1 calendar month; this is the default for each *Customer* created on Sipwise NGCP platform

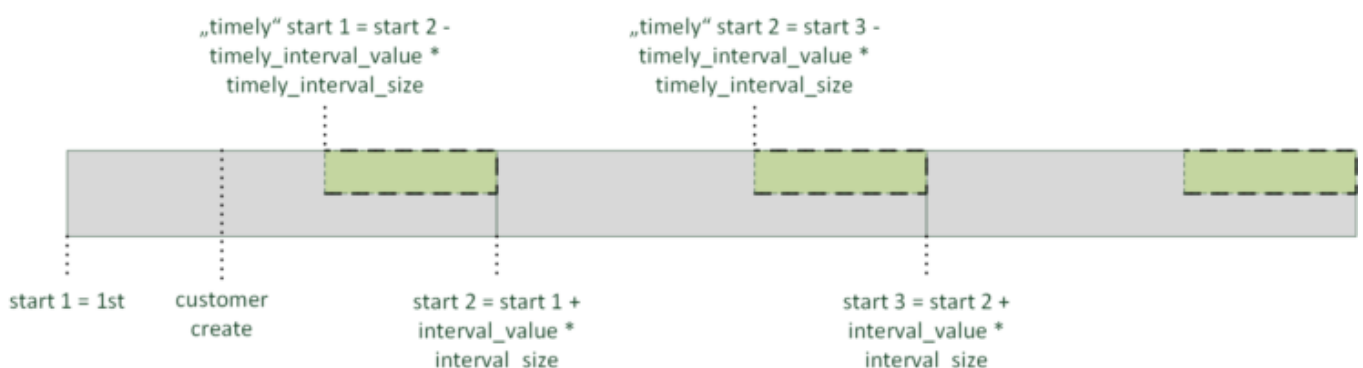


Figure 74: Interval Start Mode: 1st

- upon customer creation (create): (the initial) billing interval starts when the *Customer* is created



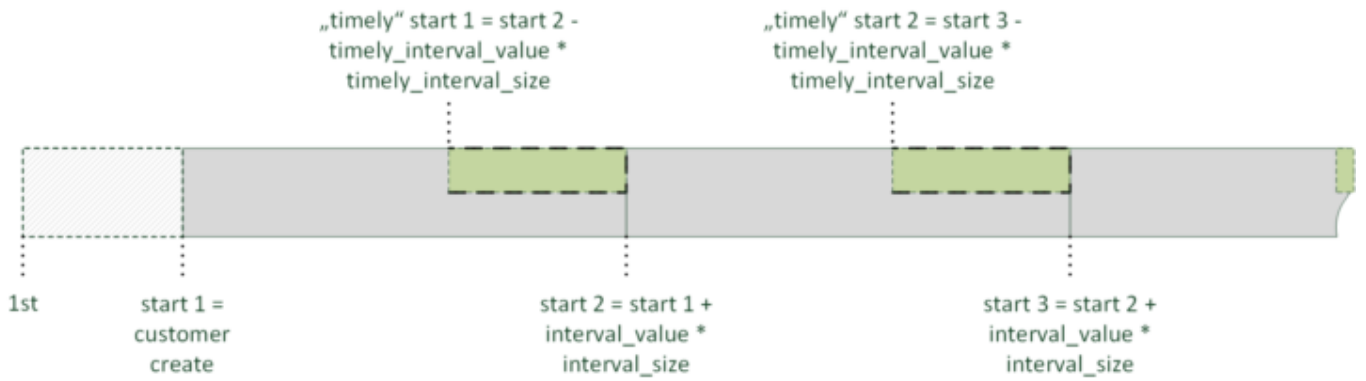


Figure 75: Interval Start Mode: create

- upon topup (topup\_interval): interval starts at *first topup* event and its length is defined by `interval duration` parameter of the profile package

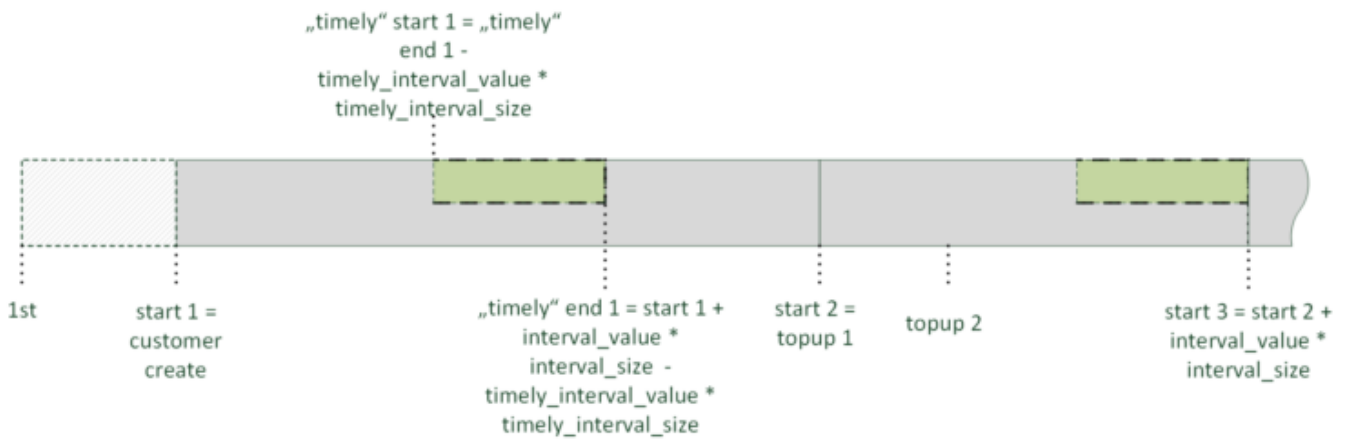


Figure 76: Interval Start Mode: topup\_interval

- intervals from topup to topup (topup): interval starts at *any topup* event and its length is defined by `interval duration` parameter of the profile package; intervals can overlap in this case

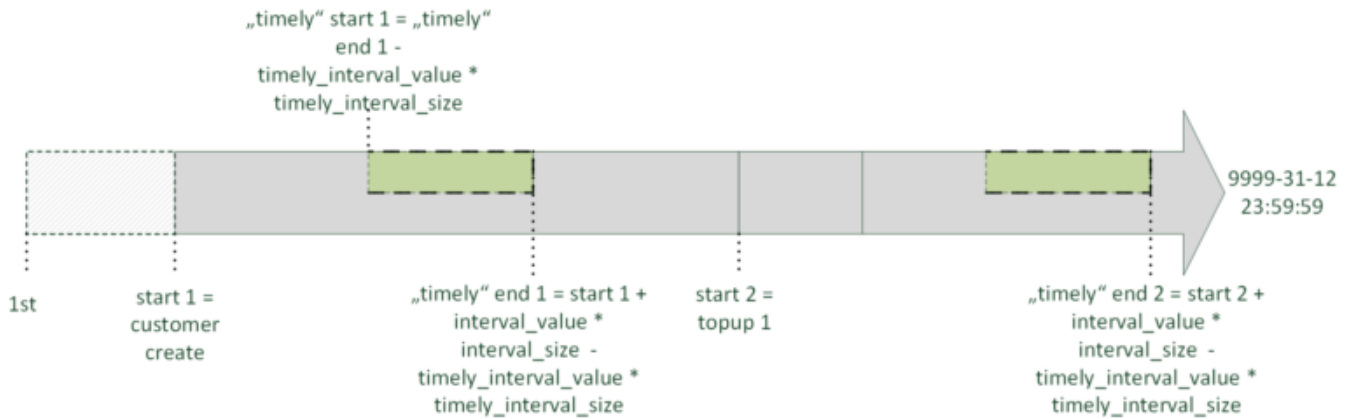


Figure 77: Interval Start Mode: topup

- Initial balance: the initial value of account balance (e.g. every new customer gets 5 Euros as a starting bonus)

### Balance Carry Over

- Carry Over: balance carry over behaviour upon interval transitions:
  - `carry-over`: always keep balance
  - `carry-over only if topped-up timely`: keep balance in case of a *timely* top-up only; where **timely** means the topup happens within a pre-defined time span before the end of the balance interval
  - `discard`: discard balance at the end of each interval
- Timely Duration: duration of the *timely* period
- Discard balance after intervals: for how many balance intervals the remaining account balance is kept before its disposal

### Underrun Settings

- Underrun lock threshold: when account balance reaches this amount the subscriber will be locked to a restricted set of services
- Underrun lock level: this level of services will apply when an account balance underruns
  - `don't change`: no change in the available set of services
  - `no lock`: all services are available
  - `foreign`: only calls within subscriber's own domain are allowed
  - `outgoing`: all outgoing calls are prohibited
  - `all calls`: all calls (incoming + outgoing) are prohibited
  - `global`: all calls + access to Customer Self Care web interface are prohibited
  - `ported`: only automatic call forwarding, due to number porting, is allowed
- Underrun profile threshold: when account balance reaches this amount the *Underrun Billing Profile* will be applied

**Basic Top-up Settings**

- Top-up lock level: subscriber lock (unlock) levels to apply upon top-up event
- Service charge: (always) subtract this value from the voucher amount, if topup happens via the usage of a voucher

**Profile mappings**

A lists of (billing profile, billing network) tuples for appending profile mappings:

- Initial Billing Profile: when creating or manually changing the customers package (initial\_profiles)
- Underrun Billing Profile: when the balance underuns a cash threshold (underrun\_profiles)
- Top-up Billing Profile: when the customer tops-up using a voucher associated with the package (topup\_profiles)

**7.5.3.2 Examples**

**Profile Package Configuration**

1. Definition of basic profile package parameters

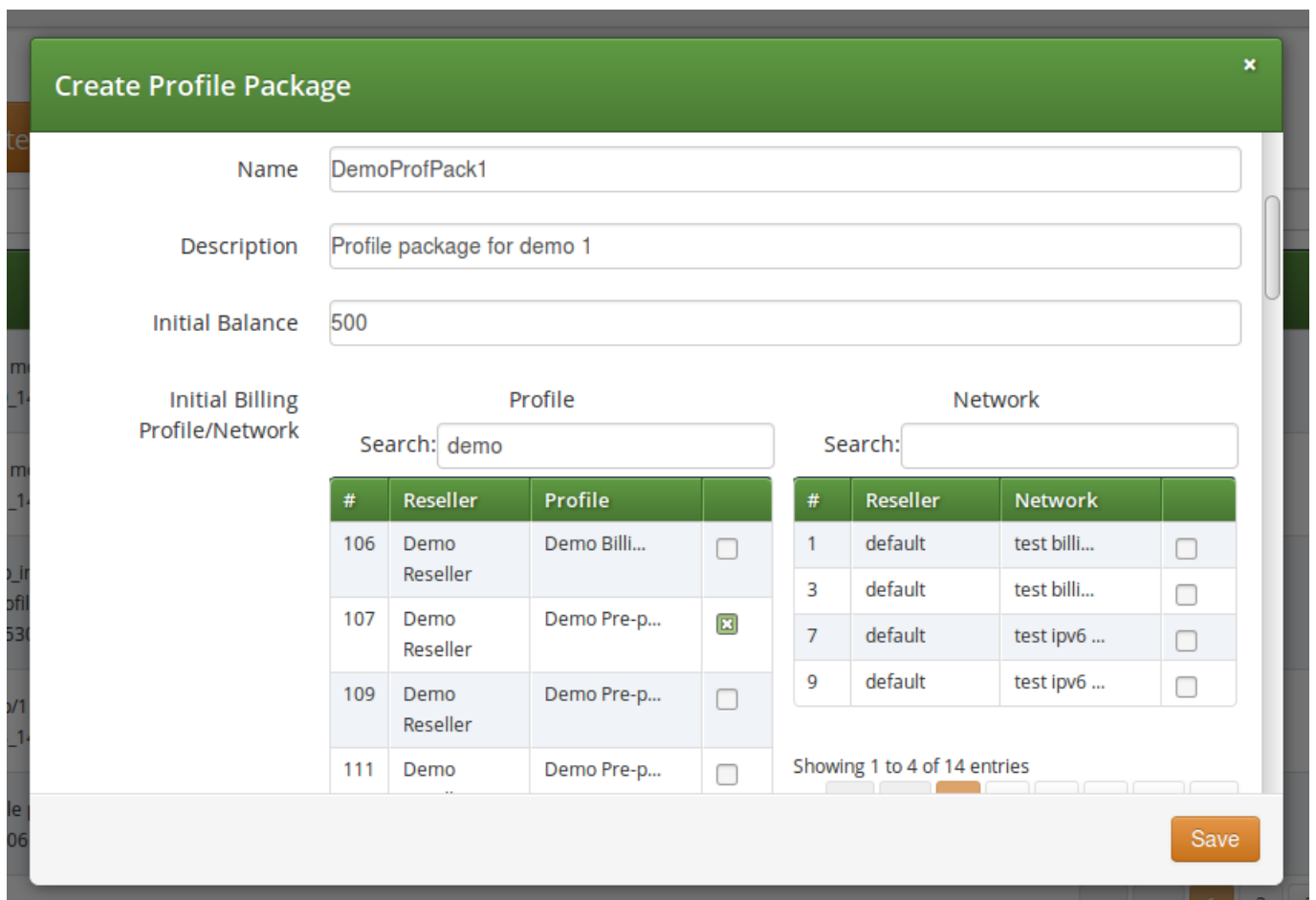


Figure 78: Basic Profile Package Parameters

2. Definition of balance interval and carry-over behaviour

The screenshot shows a 'Create Profile Package' dialog box with the following fields and values:

Field	Value
Balance Interval	1 month(s)
Balance Interval Start	1st day of month
Carry Over	carry over
"Timely" Duration	minute(s)
Discard balance after Intervals	3
Underrun lock threshold	100
Underrun lock level	outgoing
Underrun profile threshold	300

A 'Save' button is located at the bottom right of the dialog box.

Figure 79: Balance Interval and Carry-over

3. Definition of balance underrun parameters

**Create Profile Package**
✕

Underrun lock threshold

Underrun lock level

Underrun profile threshold

Underrun Billing Profile/Network

Profile

Search:

#	Reseller	Profile	
113	Demo Reseller	Demo Pre-p...	<input type="checkbox"/>
115	Demo Reseller	Demo Pre-p...	<input checked="" type="checkbox"/>
117	Demo Reseller	Demo Pre-p...	<input type="checkbox"/>

Network

Search:

#	Reseller	Network	
1	default	test billi...	<input type="checkbox"/>
3	default	test billi...	<input type="checkbox"/>
7	default	test ipv6 ...	<input type="checkbox"/>
9	default	test ipv6 ...	<input type="checkbox"/>

Figure 80: Balance Underrun Parameters

4. Definition of top-up settings

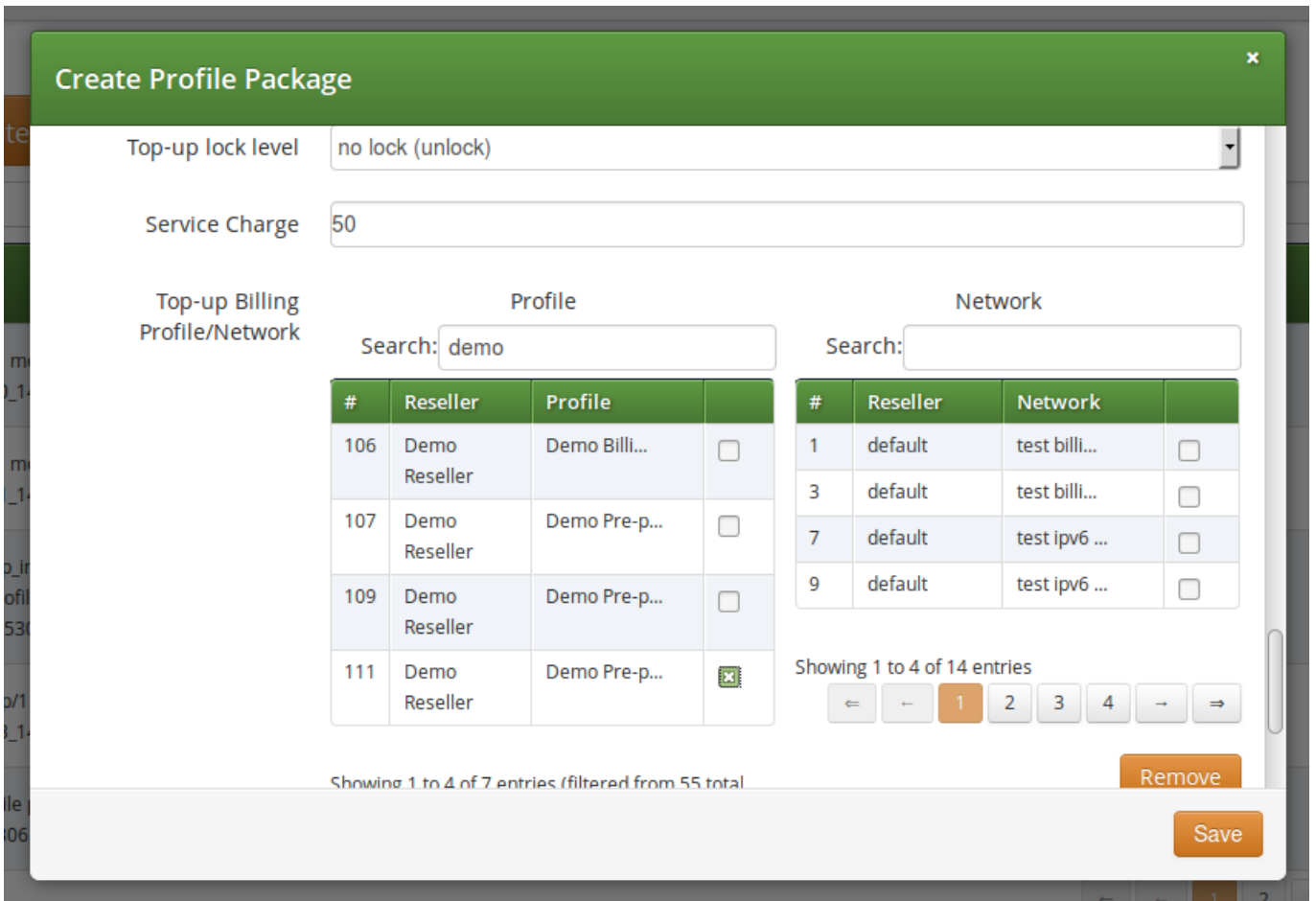


Figure 81: Balance Top-up Settings

5. Assigning a profile package to a customer

**Edit Customer #197**

Set billing profiles

Package

#	Reseller	Package	
67	Demo Reseller	DemoProfPack1	<input checked="" type="checkbox"/>
69	Demo Reseller	DemoProfpack2	<input type="checkbox"/>

Showing 1 to 2 of 2 entries (filtered from 32 total entries)

Product

#	Name	
4	Basic SIP Account	<input checked="" type="checkbox"/>
5	Cloud PBX Account	<input type="checkbox"/>

cust\_contact0@custcontact.invalid Basic SIP SILVER NETWORK Y 1473815306 active

Figure 82: Assigning Profile Package to Customer

**Interval start mode: top-up interval; carry-over: timely**

Profile package setup:

- initial\_balance: 1.0 euro
- balance\_interval: 30 "day(s)"
- interval\_start\_mode: "topup\_interval"
- carry\_over\_mode: "timely"
- timely\_duration: 12 "day(s)"
- underrun\_lock\_threshold: 0.7 euro
- underrun\_profile\_threshold: 5.0 euro
- underrun\_lock\_level:...

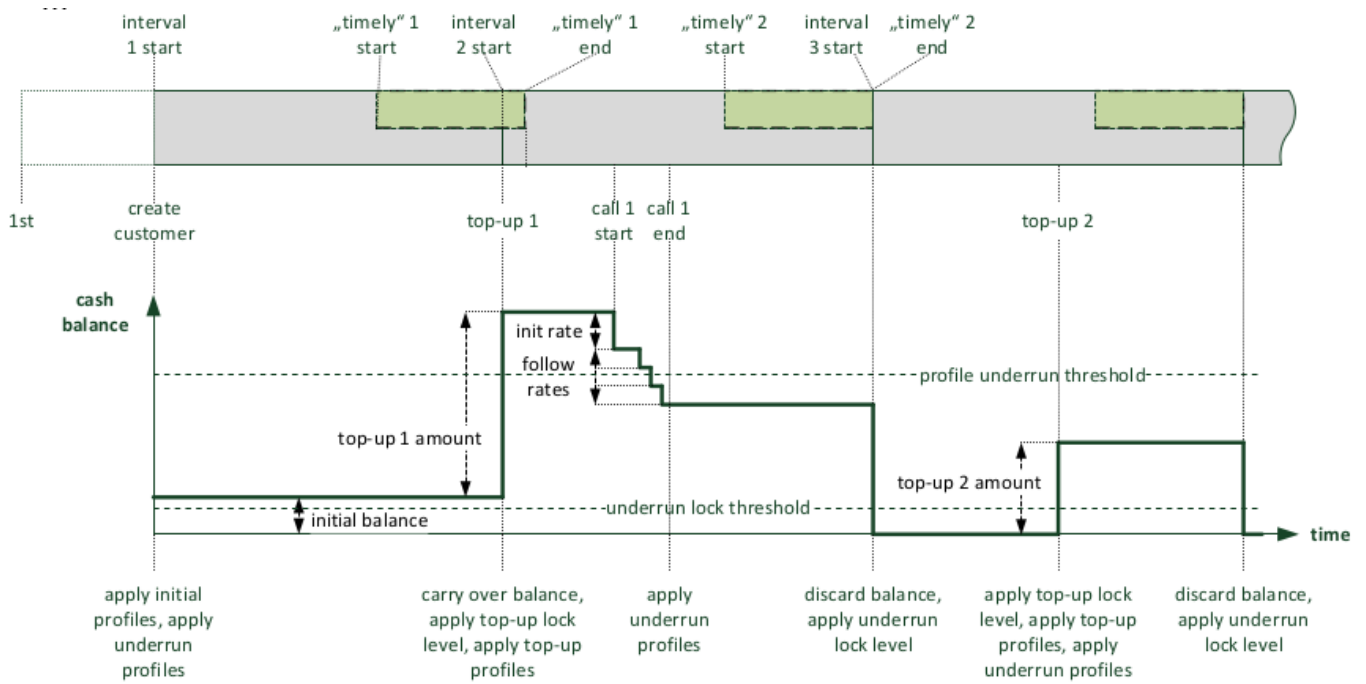


Figure 83: Example: Top-up Interval and Timely Carry-over

**Interval start mode: top-up to top-up; carry-over: always**

- initial\_balance: 1.0 euro
- balance\_interval: 30 "day(s)"
- interval\_start\_mode: "topup"
- carry\_over\_mode: "carry-over"
- notopup\_discard\_intervals: 1
- underrun\_lock\_threshold: 0.7 euro
- underrun\_profile\_threshold: 5.0 euro
- underrun\_lock\_level:...



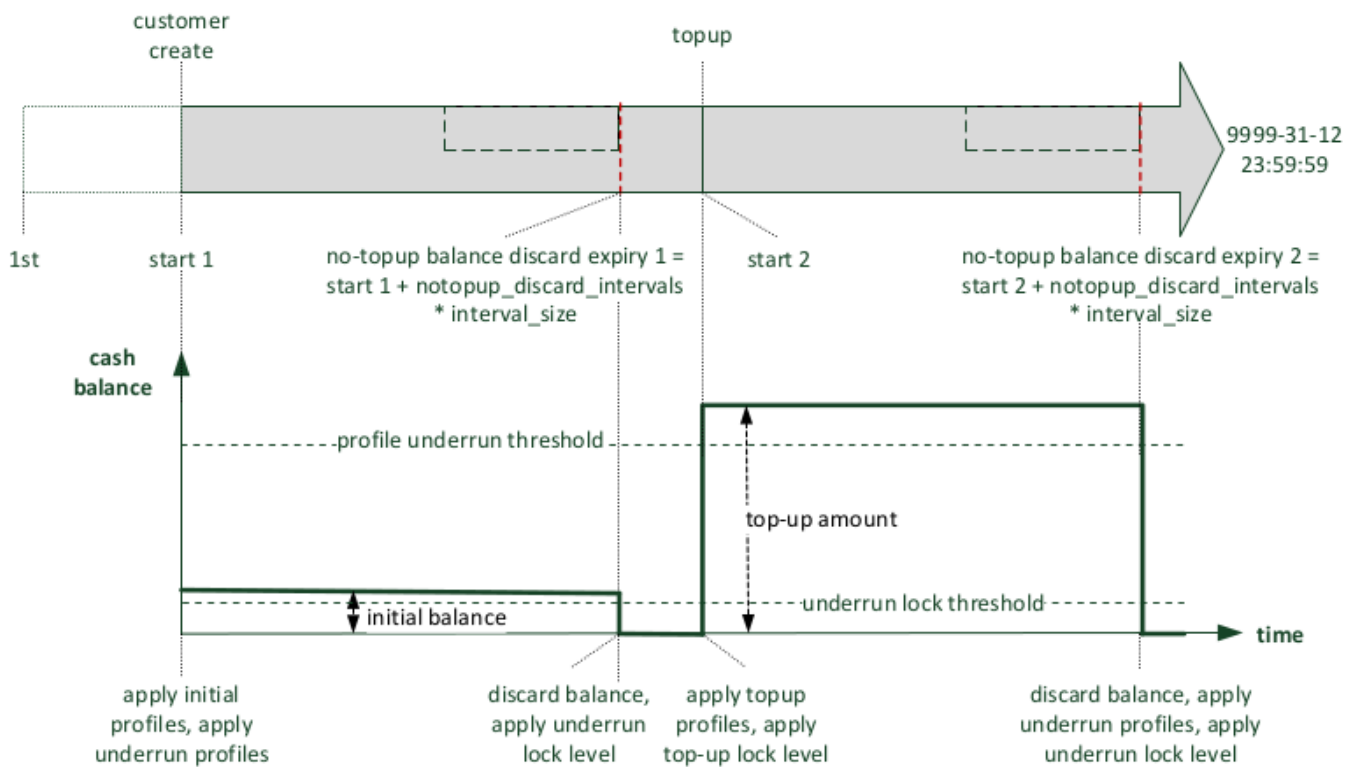


Figure 84: Example: Top-up and Always Carry-over

### 7.5.3.3 REST API

The new `/api/profilepackages/` REST API resource allows to manage billing profile package container entities, that aggregate settings of profile packages.

A sample JSON structure follows:

```
{
  "reseller_id" : 1,
  "status" : "active",
  "name" : "demo profile package",
  "description" : "package for 10€ ...",
  "balance_interval_start_mode" : "1st",
  "balance_interval_value" : 1,
  "balance_interval_unit" : "month",
  "carry_over_mode" : "carry_over",
  "timely_duration_unit" : null,
  "timely_duration_value" : null,
  "initial_balance" : 0,
  "initial_profiles" : [...], // required default, e.g. same as „topup_profiles“
  "notopup_discard_intervals" : null,
  "underrun_lock_threshold" : 0,
  "underrun_lock_level" : 4,
}
```

```

"underrun_profile_threshold" : 5,
"underrun_profiles" : [...],
"service_charge" : 10,
"topup_lock_level" : null,
"topup_profiles" : [ {
    "network_id" : null, // any network
    "profile_id" : 29
  },
  {
    "network_id" : 2, // a specific billing network
    "profile_id" : 30
  },
],
...
}

```

### 7.5.4 Vouchers

Vouchers are a typical mean of topping-up an account balance in pre-paid billing scenarios.

The definition of a voucher in the database may succeed via:

- manual entry of voucher data on the administrative web panel or through the REST API
- bulk-uploading of vouchers using a CSV (comma separated value) formatted file

In order to manage vouchers the administrator has to navigate to: *Settings* → *Vouchers* → *Create Billing Voucher* or select an existing one and press *Edit* button.

## Billing Vouchers

← Back
★ Create Billing Voucher
★ Upload Vouchers as CSV

Billing voucher successfully created

Show  entries Search:

#	Code	Amount	Reseller	Profile Package	For Contract #	Valid Until	Used At	Used By Subscriber #
25	DEMO_Voucher_Profpack1_001	1000	Demo Reseller	DemoProfPack1		2017-12-31 23:59:59		
27	DEMO_Voucher_Profpack2_001	2000	Demo Reseller	DemoProfpack2		2018-06-30 23:59:59		

Showing 1 to 2 of 2 entries (filtered from 14 total entries)

← 1 →

Figure 85: List of Vouchers

### 7.5.4.1 Properties of Vouchers

- Code: the unique code of the voucher which assures that a voucher can be used only once; this property is encrypted and displayed on the web panel to authorized users only
- Amount: the amount of money the voucher represents
- Valid until: end of validity period

**Create Billing Vouchers**

Reseller Search:

#	Name	Contract #	Status
16	Demo Reseller	200	active

Showing 1 to 1 of 1 entries (filtered from 9 total entries)

Code:

Amount:

Valid until:

Customer Search:

#	Reseller	Contact Email	External #	Status
---	----------	---------------	------------	--------

Figure 86: Voucher's Main Properties

Setting following properties of a voucher is optional:

- Customer: the *Customer* whom the voucher will be assigned to; subscribers of other customers can not redeem the voucher
- Package: vouchers may be associated with profile packages; if done so, some changes will be applied to the *Customer* for whom the voucher is redeemed with the top-up event:
  - applying top-up profile mappings starting with the time of the top-up
  - subtracting the new package's service charge from the voucher amount

- resizing the current balance interval for a gapless transition, if the new package has a different interval start mode (e.g. from "create" to "1st")
- if a new balance interval starts with the top-up, the carry-over mode of the customer's previous package applies

**Create Billing Vouchers**
✕

Customer

#	Reseller	Contact Email	External #	Status	
7	default	customer.test@spce.test		active	<input type="checkbox"/>
13	default	cust_contact0@custcontact.invalid		active	<input type="checkbox"/>
15	default	cust_contact0@custcontact.invalid		active	<input type="checkbox"/>
17	default	cust_contact0@custcontact.invalid		active	<input type="checkbox"/>

Showing 1 to 4 of 71 entries

Search:

Create Contract

Package

#	Reseller	Package	
69	Demo Reseller	DemoProfpack2	<input type="checkbox"/>
67	Demo Reseller	DemoProfPack1	<input checked="" type="checkbox"/>

Search:

Save

Figure 87: Voucher: Customer and Profile Package

#### 7.5.4.2 REST API

Vouchers can be created and managed using the `/api/vouchers/` REST API resource. This resource restricts invasive operations (POST, PUT, PATCH, DELETE) to authorized users.

```
{
  "amount" : 1000,
  "customer_id" : null, //do not restrict to a specific customer
  "valid_until" : "2017-06-05 23:59:59",
  "package_id" : "571", //switch to profile package
  "reseller_id" : 1,
  "code" : "SILVER_1_1437974823"
}
```

### 7.5.5 Top-up

A customer's administrator or subscriber can perform a top-up to increase the contract's cash balance. The NGCP platform supports two means of topping-up the balance:

1. Top-up Cash: Directly specify the cash amount to add
2. Top-up Voucher: Specify the code of a voucher, which was set up in advance

The NGCP platform provides 2 interfaces to perform top-ups:

1. through the REST API: use a CRM or third-party REST-API Broker (which i.e. coordinates with an App-Store purchase process) to finally instruct NGCP to perform a top-up. This is the **recommended** method.
2. through the administrative web interface:

One has to select the *Customer*, then *Details* → *Contract Balance* and finally press *Top-up Cash* or *Top-up Voucher*.

#### 7.5.5.1 Top-up Cash

When doing top-up with cash one needs to supply the amount of top-up in the currency of the customer contract. Optionally one can assign a *Profile Package* to the top-up event which will activate that profile package for the customer.

**Edit Settings** ✕

Amount

Package

#	Reseller	Package	
67	Demo Reseller	DemoProfPack1	<input checked="" type="checkbox"/>
69	Demo Reseller	DemoProfpack2	<input type="checkbox"/>

Showing 1 to 2 of 2 entries

← → 1 ← →

Create Profile Package

Perform top-up

Figure 88: Balance Top-up with Cash

It is also possible to perform top-up through the **REST API**: `POST /api/topupcash`

```
{
  "subscriber_id" : "73",
  "amount" : 100,
  "package_id" : null,
}
```

### 7.5.5.2 Top-up Voucher

Selecting *Top-up Voucher* option will provide a simple list of available vouchers from which the administrator can choose the voucher. If a *Profile Package* is assigned to the voucher, that package will be activated for the customer on the top-up event.

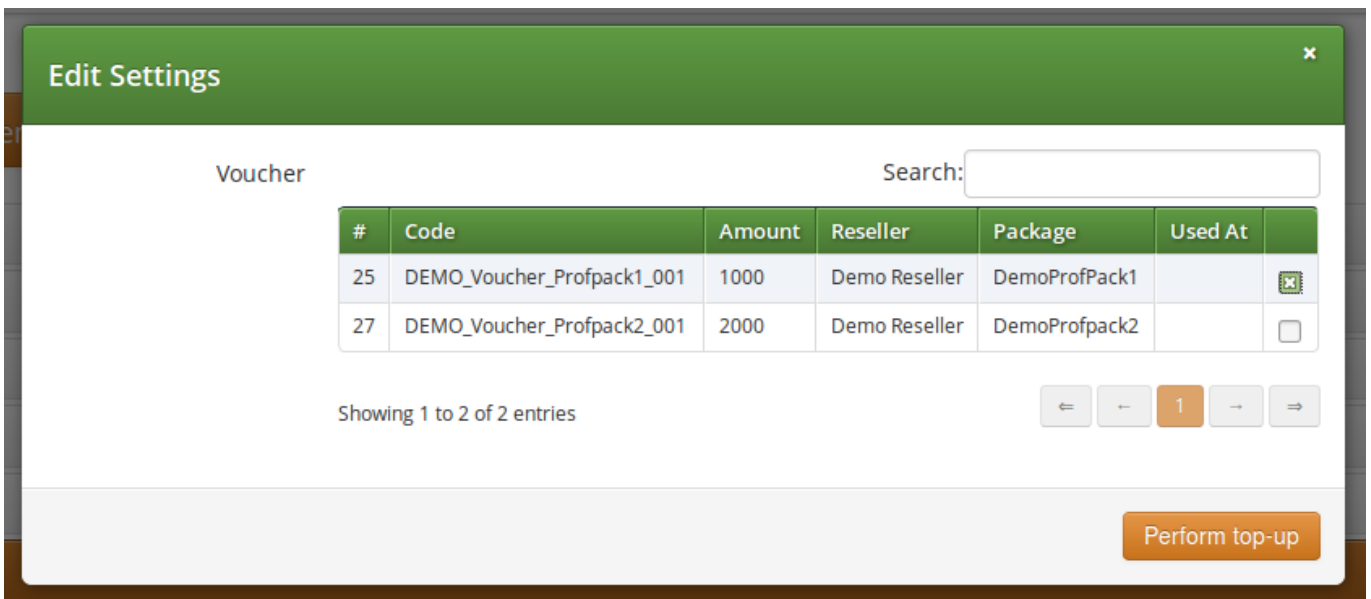


Figure 89: Balance Top-up with Voucher

It is also possible to perform top-up through the **REST API**: `POST /api/topupvouchers`

```
{
  "subscriber_id" : "73",
  "code" : "SILVER_1_1437974390"
  "request_token" : "uuid_from_3rdparty_relay" // optional request identifier
  // for lookups in the top-up log
}
```

### 7.5.6 Balance Overviews

The actual contract balance and logs of top-up or balance interval change events are a kind of financially important information and that's why those are provided on the administrative web interface for each customer. One should navigate to: *Settings* → *Customers* → *select the customer* → *Details*.

The various information details available on the web interface are discussed in subsequent sections of the handbook.

### 7.5.6.1 Contract Balance

This part of the overviews shows the actual financial state of the customer’s balance and the current profile package and balance interval.

Sound Sets

Contract Balance

↻ Top-up Voucher
↻ Top-up Cash
🔗 Set Cash Balance

Cash balance	11.50	Debit	0.00
Free time balance	0	Free time spent	0

Interval from	2016-10-01T00:00:00	Interval to	2016-10-31T23:59:59
"Timely" top-ups from		"Timely" top-ups to	
Balance will be discarded, if no top-up happens until		2017-02-01T00:00:00	

Actual profile package	DemoProfPack1	Actual billing profile	Demo Pre-paid Topup 1
Balance threshold when underrun profiles get applied	1.00	Balance threshold when subscribers will be locked	1.00

Balance Intervals

Top-up Log

Figure 90: Contract Balance Status

Another functionality assigned to *Contract Balance* section is the manual top-up. Both top-up with cash and top-up with voucher can be performed from here.

### 7.5.6.2 Balance Intervals

This table shows the balance intervals that have been in use, including the current interval.

Sound Sets								
Contract Balance								
Balance Intervals								
Show 5 entries					Search:			
From	To	Cash	Debit	#Top-ups	#Timely Top-ups	Underrun detected (Profiles)	Underrun detected (Lock)	
2016-09-01 00:00:00	2016-09-30 23:59:59	0.00	0.00	0	0			
2016-10-01 00:00:00	2016-10-31 23:59:59	11.50	0.00	1	0	2016-10-07 15:05:26	2016-10-07 15:05:26	
Showing 1 to 2 of 2 entries								<input type="button" value="←"/> <input type="button" value="→"/> <input type="button" value="1"/> <input type="button" value="↔"/>
Top-up Log								
Fraud Limits								

Figure 91: List of Balance Intervals

**Content** of the balance intervals table is:

- From, To: starting and end points of the time interval
- Cash: the contract’s cash balance value at the end of the interval (former int.), or currently (actual int.)
- Debit: the total spent amount of money in the actual interval

**Note**

While "Cash" shows the remaining amount, "Debit" shows the spent amount. With a post-paid billing scenario only "Debit" field would be populated, with pre-paid both fields will display an amount.

- No. of Top-ups: how many top-up events happened within the interval
- No. of Timely Top-ups: how many timely top-up events happened within the interval
- Underrun detected (Profiles or Lock): the time of last underrun event when either an underrun billing profile, or a subscriber lock was activated

**7.5.6.3 Top-up Log**

Each successful or failing top-up request has to be logged. The log records represent an audit trail and reflect any data changes in the course of the top-up request.

In case of an error during the top-up operation the error message and any parseable fields of failed top-up attempts is recorded.



Contract Balance											
Balance Intervals											
Top-up Log											
Show	5	entries	From Date:		To Date:		Search:				
Timestamp	Subscriber	Type	Outcome	Message	Voucher ID	Amount	Balance before	Balance after	Package before	Package after	
2016-10-07 15:11:29		cash	ok			11.50	0.00	11.50	DemoProfPack1	DemoProfPack1	
Showing 1 to 1 of 1 entries											<input type="button" value="←"/> <input type="button" value="--"/> <input type="button" value="1"/> <input type="button" value="--"/> <input type="button" value="→"/>
Fraud Limits											
Invoices											

Figure 92: Balance Top-up Log

**Content** of the top-up log table is:

- **Timestamp:** when the top-up happened
- **Subscriber:** the ID of the subscriber who performed the top-up
- **Type:** cash or voucher
- **Outcome:** ok or failed
- **Message:** error message, if Outcome="failed"
- **Voucher ID:** ID of voucher, if Type="voucher"
- **Amount:** the amount by which the balance was modified (after the *Service Charge* was subtracted from the voucher's value)
- **Balance before:** balance's value before top-up
- **Balance after:** balance's value after top-up
- **Package before:** the name of the *Profile Package* that was active before top-up
- **Package after:** the name of the *Profile Package* that became active after top-up

The top-up log table can also be queried using the readonly `/api/topuplogs` **REST API** resource.

An example of the response:

```
{
  "_embedded" : {
    "ngcp:topuplogs" : [{
      "_links" : {...},
      "amount" : null,
      "cash_balance_after" : null,
      "cash_balance_before" : null,

```

```

    "contract_balance_after_id" : null,
    "contract_balance_before_id" : null,
    "contract_id" : 2565,
    "id" : 373,
    "lock_level_after" : null,
    "lock_level_before" : null,
    "message" : ..., //error reason
    "outcome" : "failed",
    "package_after_id" : null,
    "package_before_id" : null,
    "profile_after_id" : null,
    "profile_before_id" : null,
    "request_token" : "1444956281_6", // = "panel" for panel UI requests
    "subscriber_id" : 1804,
    "timestamp" : "2015-10-16 02:45:19",
    "type" : "voucher", // "cash" or "voucher"
    "username" : "administrator",
    "voucher_id" : null }]
  },
  "_links" : { ... },
  "total_count" : 1
}

```

### 7.5.7 Usage Examples

After getting to know the concepts of customized billing solution on sip:carrier platform, it's worth seeing some practical examples for the usage of those advanced features.

The starting point is the setup of *Profile Packages* for our fictive customers: A, B and C. There are 4 different packages defined, with corresponding vouchers:

- **Initial:**

- Balance interval: 1 month
- Timely duration: 1 month
- Interval start mode: topup\_interval
- Carry-over mode: carry\_over\_timely

- **Silver:**

- Balance interval: 1 month
- Timely duration: 1 month
- Interval start mode: "topup\_interval"
- Carry-over mode: "carry\_over\_timely"
- Service charge: 2 EUR

- Underrun lock level: "no lock"
- Voucher value: 10 EUR
- **Gold:**
  - Balance interval: 1 month
  - Interval start mode: "topup\_interval"
  - Carry-over mode: "carry\_over"
  - Service charge: 5 EUR
  - Underrun lock level: "no lock"
  - Voucher value: 20 EUR
- **Extension:**
  - Balance interval: 1 month
  - Timely duration: 1 month
  - Interval start mode: "topup\_interval"
  - Carry-over mode: "carry\_over\_timely"
  - Service charge: 2 EUR
  - Underrun lock level: "no lock"
  - Voucher value: 2 EUR

### 7.5.7.1 Customer A — Silver Package

1. Customer A tops up 10 EUR with a "silver" voucher. 2 EUR are deducted as service charge. Remaining balance is 8 EUR starting on the date of the top-up.
2. Customer A doesn't top-up balance within the next month, so remaining balance is set to 0 after one month, and billing profiles and lock levels are set to the balance-underrun definition of the "silver" package.

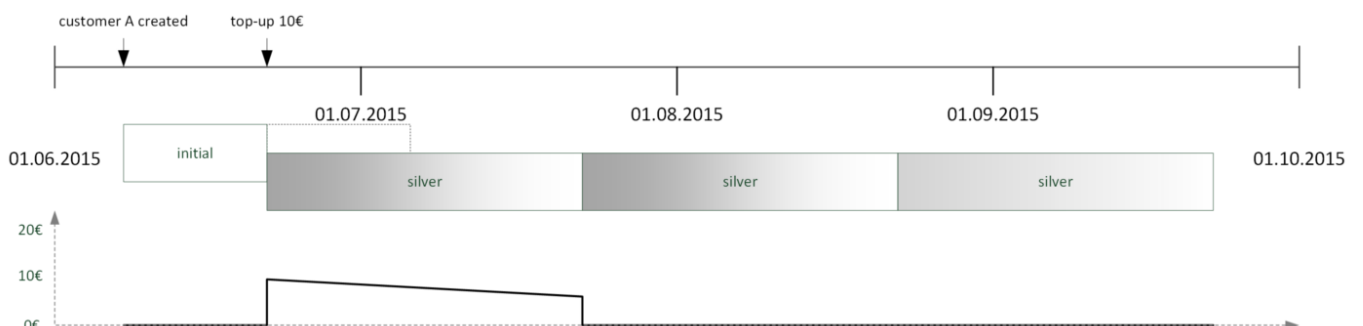


Figure 93: Usage Example: Silver Package

### 7.5.7.2 Customer B — Silver and Extension Package

1. Customer B tops up 10 EUR with the “silver” voucher. 2 EUR are deducted as service charge. Remaining balance is 8 EUR starting on the date of the top-up.
2. Customer B tops up 2 EUR using an “extension” voucher on the last day. 2 EUR are deducted as service charge and the interval is extended for one month, carrying over his old balance.
3. Customer B doesn’t top-up balance within the next month, so remaining balance is set to 0 after the month, and billing profiles and lock levels are set to the balance-underrun definition of the “extension” package.

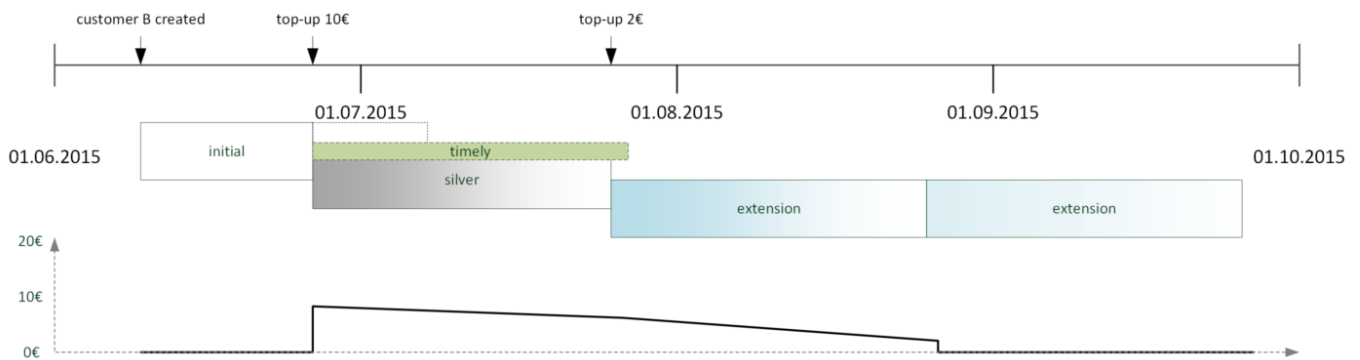


Figure 94: Usage Example: Silver + Extension Package

### 7.5.7.3 Customer C — Gold Package

Customer C tops up 20 EUR with the “gold” voucher. 5 EUR are deducted as service charge. Remaining balance is 15 EUR starting on the date of the top-up. Balance is carried over after each month until used up.

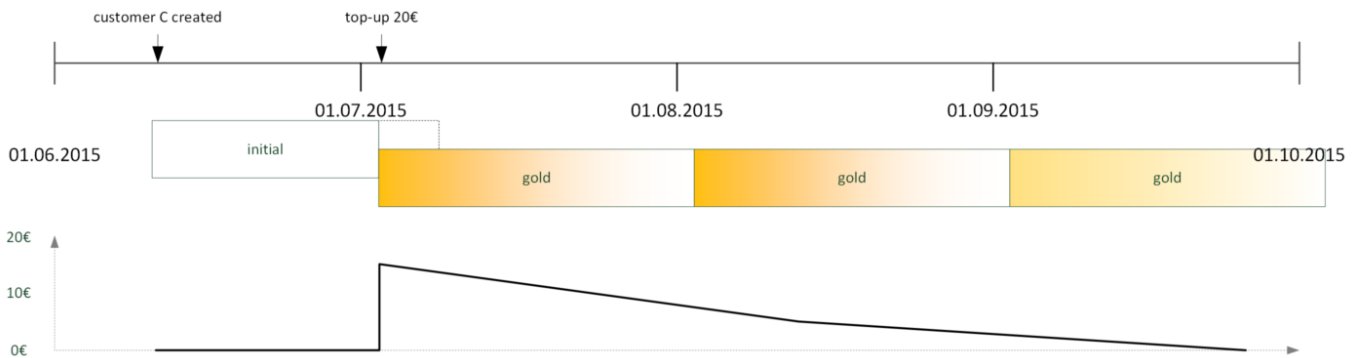


Figure 95: Usage Example: Gold Package

## 7.6 Billing Data Export

Regular billing data export is done using CSV (*comma separated values*) files which may be downloaded from the platform using the *cdrexpert* user which has been created during the installation.

There are two types of exports. One is *CDR* (Call Detail Records) used to charge for calls made by subscribers, and the other is *EDR* (Event Detail Records) used to charge for provisioning events like enabling certain features.

### 7.6.1 Glossary of Terms

Billing records contain fields that hold data of various entities that play a role in the phone service offered by Sipwise NGCP. For a better understanding of billing data please refer to the glossary provided here:

- **Account:** the customer's account that is charged for calls of its subscriber(s)
- **Carrier:** a SIP peer that sends incoming calls to, or receives outgoing calls from NGCP. A carrier may charge fees for the outgoing calls from NGCP (outbound billing fee), or for the incoming calls to NGCP (inbound billing fee).
- **Contract:** the service contract that represents a customer, a reseller or a SIP peer; a contract on NGCP contains the billing profile (billing fees) too
- **Customer:** the legal entity that represents any number of subscribers; this entity receives the bills for calls of its subscriber(s)
- **Provider:** either the reseller that holds a subscriber who is registered on NGCP, or the SIP peer that handles calls between an external subscriber and NGCP
- **Reseller:** the entity who is the direct, administrative service provider of a group of customers and subscribers registered on NGCP; the NGCP operator may also charge a reseller for the calls initiated or received by its subscribers
- **User:** the subscriber who either is registered on NGCP, or is an external call party

### 7.6.2 File Name Format

In order to be able to easily identify billing files, the file names are constructed by the following fixed-length fields:

```
<prefix><separator><version><separator><timestamp><separator><sequence number>< ←
  suffix>
```

The definition of the specific fields is as follows:

Table 13: CDR/EDR export file name format

File name element	Length	Description
<prefix>	7	A fixed string. Always sipwise.
<separator>	1	A fixed character. Always _.
<version>	3	The format version, a three digit number. Currently 007.
<timestamp>	14	The file creation timestamp in the format YYYYMMDDhhmmss.
<sequence number>	10	A unique 10-digit zero-padded sequence number for quick identification.
<suffix>	4	A fixed string. Always .cdr or .edr.

A valid example filename for a CDR billing file created at 2012-03-10 14:30:00 and being the 42nd file exported by the system, is:  
 sipwise\_007\_20130310143000\_0000000042.cdr

### 7.6.3 File Format

Each billing file consists of three parts: one header line, zero to 5000 body lines and one trailer line.

#### 7.6.3.1 File Header Format

The billing file header is one single line, which is constructed by the following fields:

```
<version>,<number of records>
```

The definition of the specific fields is as follows:

Table 14: CDR/EDR export file header line format

Body Element	Length	Type	Description
<version>	3	zero-padded uint	The format version. Currently 007.
<number of records>	4	zero-padded uint	The number of body lines contained in the file.

A valid example for a Header is:

```
007,0738
```

#### 7.6.3.2 File Body Format for Call Detail Records (CDR)

The body of a CDR consists of a minimum of zero and a default maximum of 5000 lines. The platform operator can configure the maximum number of lines kept in a file by updating the `cdrexport.max_rows_per_file` parameter in `/etc/ngcp-config/config.yml` file. Each line holds one call detail record in CSV format and is constructed by a configurable set of fields, all of them enclosed in single quotes.

The following table defines the **default set of fields** that are inserted into the CDR file, for exports related to *system* scope. The list of fields is defined in `/etc/ngcp-config/config.yml` file, `cdrexport.admin_export_fields` parameter.

Table 15: Default set of system CDR fields

Body Element	Length	Type	Description
CDR_ID	1-10	uint	Internal CDR ID.
UPDATE_TIME	19	timestamp	Timestamp of last modification, including date and time (with seconds precision).
SOURCE_USER_ID	36	string	Internal UUID of calling party subscriber. Value is 0 if calling party is external.
SOURCE_PROVIDER_ID	0-255	string	Internal ID of the contract of calling party provider (i.e. reseller or peer).
SOURCE_EXTERNAL_SUBSCRIBER_ID	0-255	string	External, arbitrary ID of calling party subscriber. (A string value shown as "External ID" property of an NGCP subscriber.)
SOURCE_SUBSCRIBER_ID	1-11	uint	Internal ID of calling party subscriber. Value is 0 if calling party is external.
SOURCE_EXTERNAL_CONTRACT_ID	0-255	string	External, arbitrary ID of calling party customer. (A string value shown as "External ID" property of an NGCP customer/peer.)
SOURCE_ACCOUNT_ID	1-11	uint	Internal ID of calling party customer.
SOURCE_USER	0-255	string	SIP username of calling party.
SOURCE_DOMAIN	0-255	string	SIP domain of calling party.
SOURCE_CLI	0-64	string	CLI of calling party in E.164 format.
SOURCE_CLIR	1	uint	1 for calls with CLIR, 0 otherwise.
SOURCE_IP	0-64	string	IP Address of the calling party.
DESTINATION_USER_ID	36	string	Internal UUID of called party subscriber. Value is 0 if called party is external.
DESTINATION_PROVIDER_ID	0-255	string	Internal ID of the contract of called party provider (i.e. reseller or peer).
DESTINATION_EXTERNAL_SUBSCRIBER_ID	0-255	string	External, arbitrary ID of called party subscriber. (A string value shown as "External ID" property of an NGCP subscriber.)
DESTINATION_SUBSCRIBER_ID	1-11	uint	Internal ID of called party subscriber. Value is 0 if calling party is external.
DESTINATION_EXTERNAL_CONTRACT_ID	0-255	string	External, arbitrary ID of called party customer. (A string value shown as "External ID" property of an NGCP customer/peer.)
DESTINATION_ACCOUNT_ID	1-11	uint	Internal ID of called party customer.

Table 15: (continued)

Body Element	Length	Type	Description
DESTINATION_USER	0-255	string	Final SIP username of called party.
DESTINATION_DOMAIN	0-255	string	Final SIP domain of called party.
DESTINATION_USER_IN	0-255	string	Incoming SIP username of called party, after applying inbound rewrite rules.
DESTINATION_DOMAIN_IN	0-255	string	Incoming SIP domain of called party, after applying inbound rewrite rules.
DESTINATION_USER_DIALED	0-255	string	The user-part of the SIP Request URI as received by NGCP.
PEER_AUTH_USER	0-255	string	Username used to authenticate towards peer.
PEER_AUTH_REALM	0-255	string	Realm used to authenticate towards peer.
CALL_TYPE	3-4	string	The type of the call - one of: call: normal call cfu: call forward unconditional cfb: call forward busy cft: call forward timeout cfna: call forward not available cfs: call forward for SMS
CALL_STATUS	2-8	string	The final call status - one of: ok: successful call busy: called party busy noanswer: no answer from called party cancel: cancel from caller offline: called party offline timeout: no reply from called party other: unspecified, see CALL_CODE field for details
CALL_CODE	3	string	The final SIP status code.
INIT_TIME	23	timestamp	Timestamp of call initiation (SIP <i>INVITE</i> received from calling party). Includes date, time with milliseconds (3 decimals).
START_TIME	23	timestamp	Timestamp of call establishment (final SIP response received from called party). Includes date, time with milliseconds (3 decimals).
DURATION	4-13	fixed precision (3 decimals)	Length of call (calculated from START_TIME) including milliseconds (3 decimals).



Table 15: (continued)

Body Element	Length	Type	Description
CALL_ID	0-255	string	The SIP Call-ID.
RATING_STATUS	2-7	string	The internal rating status of the CDR - one of: unrated: not rated ok: successfully rated failed: error while rating Currently always ok or unrated, depending on whether rating is enabled or not.
RATED_AT	0-19	datetime	Time of rating, including date and time (with seconds precision). Empty if CDR is not rated.
SOURCE_CARRIER_COST	7-14	fixed precision (6 decimals)	The originating carrier cost that the carrier (i.e. SIP peer) charges for the calls routed to his network, or empty if CDR is not rated. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>
SOURCE_CUSTOMER_COST	7-14	fixed precision (6 decimals)	The originating customer cost, or empty if CDR is not rated.
SOURCE_CARRIER_ZONE	0-127	string	Name of the originating carrier billing zone, or onnet if data is not available. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>
SOURCE_CUSTOMER_ZONE	0-127	string	Name of the originating customer billing zone, or empty if CDR is not rated.
SOURCE_CARRIER_DETAIL	0-127	string	Description of the originating carrier billing zone, or platform internal if data is not available. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>
SOURCE_CUSTOMER_DETAIL	0-127	string	Description of the originating customer billing zone, or empty if CDR is not rated.
SOURCE_CARRIER_FREE_TIME	1-10	uint	The number of free time seconds used on originating carrier side, or empty if CDR is not rated. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>

Table 15: (continued)

Body Element	Length	Type	Description
SOURCE_CUSTOMER_FREE_TIME	1-10	uint	The number of free time seconds used from the originating customer's account balance, or empty if CDR is not rated.
DESTINATION_CARRIER_COST	7-14	fixed precision (6 decimals)	The terminating carrier cost, or empty if CDR is not rated. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>
DESTINATION_CUSTOMER_COST	7-14	fixed precision (6 decimals)	The terminating customer cost, or empty if CDR is not rated.
DESTINATION_CARRIER_ZONE	0-127	string	Name of the terminating carrier billing zone, or onnet if data is not available. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>
DESTINATION_CUSTOMER_ZONE	0-127	string	Name of the terminating customer billing zone, or empty if CDR is not rated.
DESTINATION_CARRIER_DETAIL	0-127	string	Description of the terminating carrier billing zone, or empty if CDR is not rated. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>
DESTINATION_CUSTOMER_DETAIL	0-127	string	Description of the terminating customer billing zone, or empty if CDR is not rated.
DESTINATION_CARRIER_FREE_TIME	1-10	uint	The number of free time seconds used on terminating carrier side, or empty if CDR is not rated. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>
DESTINATION_CUSTOMER_FREE_TIME	1-10	uint	The number of free time seconds used from the terminating customer's account balance, or empty if CDR is not rated.
SOURCE_RESELLER_COST	7-14	fixed precision (6 decimals)	The originating reseller cost, or empty if CDR is not rated. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>

Table 15: (continued)

Body Element	Length	Type	Description
SOURCE_RESELLER_ZONE	0-127	string	Name of the originating reseller billing zone, or empty if CDR is not rated. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>
SOURCE_RESELLER_DETAIL	0-127	string	Description of the originating reseller billing zone, or empty if CDR is not rated. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>
SOURCE_RESELLER_FREE_TIME	1-10	uint	The number of free time seconds used from the originating reseller's account balance, or empty if CDR is not rated. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>
DESTINATION_RESELLER_COST	7-14	fixed precision (6 decimals)	The terminating reseller cost, or empty if CDR is not rated. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>
DESTINATION_RESELLER_ZONE	0-127	string	Name of the terminating reseller billing zone, or empty if CDR is not rated. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>
DESTINATION_RESELLER_DETAIL	0-127	string	Description of the terminating reseller billing zone, or empty if CDR is not rated. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>
DESTINATION_RESELLER_FREE_TIME	1-10	uint	The number of free time seconds used from the terminating reseller's account balance, or empty if CDR is not rated. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>
<line_terminator>	1	string	Always \n (special char LF - ASCII 0x0A).

A valid example of one body line of a rated CDR is (line breaks added for clarity):

```
'15','2013-03-26 22:09:11','a84508a8-d256-4c80-a84e-820099a827b0','1','','1','','2','testuser1','192.168.51.133','4311001','0','192.168.51.1',
```

```
'94d85b63-8f4b-43f0-b3b0-221c9e3373f2','1','','3','','4','testuser3',
'192.168.51.133','testuser3','192.168.51.133','testuser3','','','call','ok','200',
'2013-03-25 20:24:50.890','2013-03-25 20:24:51.460','10.880','44449842',
'ok','2013-03-25 20:25:27','0.00','24.00','onnet','testzone','platform internal',
'testzone','0','0','0.00','200.00','','foo','','foo','0','0',
'0.00','','','0','0.00','','','0'
```

The format of the **CDR export files generated for resellers** (as opposed to the complete system-wide export) is identical except for a few missing fields.

---

#### Note

Please check the description of fields in the table above, in order to see which fields are omitted for *reseller* related CDR exports.

---

The list of fields for *reseller* CDR export is defined in `/etc/ngcp-config/config.yml` file, `cdrexport.reseller_export_fields` parameter.

### 7.6.3.3 Extra fields that can be exported to CDRs

#### Supplementary Data

There are fields in CDR database that contain **supplementary data** related to subscribers. This data is not used by NGCP for CDR processing but rather provides the system administrator with a possibility to include supplementary information in CDRs.

---

#### Note

*This informational section is meant for problem solving / debugging purpose:* The supplementary data listed in following table is stored in `provisioning.voip_preferences` database table.

---

Table 16: Supplementary data in CDR fields

Body Element	Length	Type	Description
SOURCE_GPP0	0-255	string	Supplementary data field 0 of calling party.
SOURCE_GPP1	0-255	string	Supplementary data field 1 of calling party.
SOURCE_GPP2	0-255	string	Supplementary data field 2 of calling party.
SOURCE_GPP3	0-255	string	Supplementary data field 3 of calling party.
SOURCE_GPP4	0-255	string	Supplementary data field 4 of calling party.
SOURCE_GPP5	0-255	string	Supplementary data field 5 of calling party.
SOURCE_GPP6	0-255	string	Supplementary data field 6 of calling party.
SOURCE_GPP7	0-255	string	Supplementary data field 7 of calling party.
SOURCE_GPP8	0-255	string	Supplementary data field 8 of calling party.
SOURCE_GPP9	0-255	string	Supplementary data field 9 of calling party.
DESTINATION_GPP0	0-255	string	Supplementary data field 0 of called party.

Table 16: (continued)

Body Element	Length	Type	Description
DESTINATION_GPP1	0-255	string	Supplementary data field 1 of called party.
DESTINATION_GPP2	0-255	string	Supplementary data field 2 of called party.
DESTINATION_GPP3	0-255	string	Supplementary data field 3 of called party.
DESTINATION_GPP4	0-255	string	Supplementary data field 4 of called party.
DESTINATION_GPP5	0-255	string	Supplementary data field 5 of called party.
DESTINATION_GPP6	0-255	string	Supplementary data field 6 of called party.
DESTINATION_GPP7	0-255	string	Supplementary data field 7 of called party.
DESTINATION_GPP8	0-255	string	Supplementary data field 8 of called party.
DESTINATION_GPP9	0-255	string	Supplementary data field 9 of called party.

### **Account balance details (prepaid calls)**

There are fields in CDR database that show **changes in cash or free time balance**. In addition to that, a history of billing packages / profiles may also be present, since the NGCP vouchers, that are used to top-up, may also be set up to cause a transition of profile packages. (Which in turn can result in changing the billing profile/applicable fees). Therefore the billing package and profile valid at the time of the CDR are recorded and exposed as fields for CDR export.

---

#### **Tip**

Such fields may also be required to integrate sip:carrier with legacy billing systems.

---

#### **Note**

Please be aware that pre-paid billing functionality is only available in Sipwise *sip:provider PRO* and *sip:carrier* products.

---

The name of CDR data field consists of the elements listed below:

1. `source|destination`: decides if the data refers to calling (*source*) or called (*destination*) party
2. `carrier|reseller|customer`: the account owner, whose billing data is referred
3. `data type`:
  - A. `cash_balance|free_time_balance _ before|after`: cash balance or free time balance, before or after the call
  - B. `profile_package_id|contract_balance_id`: internal ID of the active pre-paid billing profile or the account balance

Examples:

- `source_customer_cash_balance_before`

- destination\_customer\_profile\_package\_id



#### Important

For calls spanning multiple balance intervals, the latter one will be selected, that is the balance interval where the call ended.



#### Important

There are some limitations in rating **pre-paid** calls, please visit [Pre-paid Billing](#) Section 7.3 section for details.

### 7.6.3.4 Distinguish between on-net and off-net calls CDRs

On-net calls (made only between devices on your network) are sometimes treated differently from off-net calls (terminated to or received from a peer) in external billing systems.

To distinguish between on-net and off-net calls in such a billing systems, check the **source\_user\_id** and **destination\_user\_id** fields. For on-net calls, both fields will have a different from zero value (actually, a UUID).

### 7.6.3.5 File Body Format for Event Detail Records (EDR)

The body of an EDR consists of a minimum of zero and a maximum of 5000 lines. The platform operator can configure the maximum number of lines kept in a file by updating the `eventexport.max_rows_per_file` parameter in `/etc/ngcp-config/config.yml` file. Each line holds one call detail record in CSV format and is constructed by the fields as per the subsequent table.

The following table defines the **default set of fields** that are inserted into the EDR file, for exports related to *system* scope. The list of fields is defined in `/etc/ngcp-config/config.yml` file, `eventexport.admin_export_fields` parameter.

Table 17: Default set of system EDR fields

Body Element	Length	Type	Description
EVENT_ID	1-11	uint	Internal EDR ID.

Table 17: (continued)

Body Element	Length	Type	Description
TYPE	0-255	string	The type of the event - one of: <i>start_profile</i> : A subscriber profile has been newly assigned to a subscriber. <i>end_profile</i> : A subscriber profile has been removed from a subscriber. <i>update_profile</i> : A subscriber profile has been changed for a subscriber. <i>start_huntgroup</i> : A subscriber has been provisioned as PBX / hunting group. <i>end_huntgroup</i> : A subscriber has been deprovisioned as PBX / hunting group. <i>start_ivr</i> : A subscriber has a new call-forward to Auto-Attendant. <i>end_ivr</i> : A subscriber has removed a call-forward to Auto-Attendant.
CONTRACT_EXTERNAL_ID	0-255	string	The external ID of the customer. (A string value shown as "External ID" property of an NGCP customer.)
COMPANY	0-127	string	The company name of the customer's contact.
SUBSCRIBER_EXTERNAL_ID	0-255	string	The external ID of the subscriber. (A string value shown as "External ID" property of an NGCP subscriber.) <i>PLEASE NOTE: This field is empty in case of start_huntgroup and end_huntgroup events.</i>
PILOT_PRIMARY_NUMBER	0-64	string	The pilot subscriber's primary number (HPBX subscribers). <i>PLEASE NOTE: This is not included in default set of EDR fields from NGCP version mr5.0 upwards.</i>
PRIMARY_NUMBER	0-64	string	The VoIP number of the subscriber with the highest ID (DID or primary number).
OLD_PROFILE_NAME	0-255	string	The old status of the event. Depending on the <i>event_type</i> : <i>start_profile</i> : Empty. <i>end_profile</i> : The name of the subscriber profile which got removed from the subscriber. <i>update_profile</i> : The name of the former subscriber profile which got updated. <i>start_huntgroup</i> : Empty. <i>end_huntgroup</i> : Empty. <i>start_ivr</i> : Empty. <i>end_ivr</i> : Empty.

Table 17: (continued)

Body Element	Length	Type	Description
NEW_PROFILE_NAME	0-255	string	The new status of the event. Depending on the event_type: start_profile: The name of the subscriber profile which got assigned to the subscriber. end_profile: Empty. update_profile: The name of the new subscriber profile which got applied. start_huntgroup: Empty. end_huntgroup: Empty. start_ivr: Empty. end_ivr: Empty.
TIMESTAMP	23	timestamp	Timestamp of event. Includes date, time with milliseconds (3 decimals).
RESELLER_ID	1-11	uint	Internal ID of the reseller which the event belongs to. <i>PLEASE NOTE: Only available in system exports, not for resellers.</i>
<line_terminator>	1	string	A fixed character. Always \n (special char LF - ASCII 0x0A).

A valid example of one body line of an EDR is (line breaks added for clarity):

```
"1", "start_profile", "sipwise_ext_customer_id_4", "Sipwise GmbH",
"sipwise_ext_subscriber_id_44", "436667778", "", "1", "2014-06-19 11:34:31", "1"
```

The format of the **EDR export files generated for resellers** (as opposed to the complete system-wide export) is identical except for a few missing fields.

---

#### Note

Please check the description of fields in the table above, in order to see which fields are omitted for *reseller* related EDR exports.

---

The list of fields for *reseller* EDR export is defined in `/etc/ngcp-config/config.yml` file, `eventexport.reseller_export_fields` parameter.

#### 7.6.3.6 Extra fields that can be exported to EDRs

There are fields in EDR database that contain **supplementary data** related to subscribers, for example subscriber phone numbers are such data.



Table 18: Supplementary data in EDR fields

Body Element	Length	Type	Description
SUBSCRIBER_PROFILE_SET_NAME	0-255	string	The subscriber's profile set name.
PILOT_SUBSCRIBER_PROFILE_SET_NAME	0-255	string	The profile set name of the subscriber's pilot subscriber.
PILOT_SUBSCRIBER_PROFILE_NAME	0-255	string	The profile name of the subscriber's pilot subscriber.
FIRST_NON_PRIMARY_ALIAS_USERNAME_BEFORE	0-255	string	The subscriber's non-primary alias with lowest ID, before number updates during the operation.
FIRST_NON_PRIMARY_ALIAS_USERNAME_AFTER	0-255	string	The subscriber's non-primary alias with lowest ID, after number updates during the operation.
PILOT_FIRST_NON_PRIMARY_ALIAS_USERNAME_BEFORE	0-255	string	The non-primary alias with lowest ID of the subscriber's pilot subscriber, before number updates during the operation.
PILOT_FIRST_NON_PRIMARY_ALIAS_USERNAME_AFTER	0-255	string	The non-primary alias with lowest ID of the subscriber's pilot subscriber, after number updates during the operation.
NON_PRIMARY_ALIAS_USERNAME	0-255	string	The non-primary alias of a subscriber affected by an <code>update_profile</code> , <code>start_profile</code> or <code>end_profile</code> event to track number changes.
PRIMARY_ALIAS_USERNAME_BEFORE	0-255	string	The subscriber's primary alias, before number updates during the operation.
PRIMARY_ALIAS_USERNAME_AFTER	0-255	string	The subscriber's primary alias, after number updates during the operation.
PILOT_PRIMARY_ALIAS_USERNAME_BEFORE	0-255	string	The primary alias of the subscriber's pilot subscriber, before number updates during the operation.
PILOT_PRIMARY_ALIAS_USERNAME_AFTER	0-255	string	The primary alias of the subscriber's pilot subscriber, after number updates during the operation.
FIRST_NON_PRIMARY_ALIAS_USERNAME_BEFORE_AFTER	0-255	string	Equals <code>FIRST_NON_PRIMARY_ALIAS_USERNAME_BEFORE</code> , if the value is not NULL, otherwise it's the same as <code>FIRST_NON_PRIMARY_ALIAS_USERNAME_AFTER</code> .
PILOT_FIRST_NON_PRIMARY_ALIAS_USERNAME_BEFORE_AFTER	0-255	string	Equals <code>PILOT_FIRST_NON_PRIMARY_ALIAS_USERNAME_BEFORE</code> , if the value is not NULL, otherwise it's the same as <code>PILOT_FIRST_NON_PRIMARY_ALIAS_USERNAME_AFTER</code> .

### 7.6.3.7 File Trailer Format

The billing file trailer is one single line, which is constructed by the following fields:

```
<md5 sum>
```

The `<md5 sum>` is a 32 character hexadecimal MD5 hash of the *Header* and *Body*.

To validate the billing file, one must remove the Trailer before computing the MD5 sum of the file. An example bash script to validate the integrity of the file is given below:

```
#!/bin/sh

error() { echo $@; exit 1; }
test -n "$1" || error "Usage: $0 <cdr-file>"
test -f "$1" || error "File '$1' not found"

TMPFILE="/tmp/${basename "$1"}.${$.}"
MD5=$(sed -rn '$ s/^(([a-z0-9]{32}).*\$/\1/i p' "$1") $TMPFILE"
sed '$d' "$1" > "$TMPFILE"
echo "$MD5" | md5sum -c -
rm -f "$TMPFILE"
```

Given the script is located in `cdr-md5.sh` and the CDR-file is `sipwise_001_20071110123000_0000000004.cdr`, the output of the integrity check for an intact CDR file would be:

```
$ ./cdr-md5.sh sipwise_001_20071110123000_0000000004.cdr
/tmp/sipwise_001_20071110123000_0000000004.cdr: OK
```

If the file has been altered during transmission, the output of the integrity check would be:

```
$ ./cdr-md5.sh sipwise_001_20071110123000_0000000004.cdr
/tmp/sipwise_001_20071110123000_0000000004.cdr: FAILED
md5sum: WARNING: 1 of 1 computed checksum did NOT match
```

### 7.6.4 File Transfer

Billing files are created twice per hour at minutes 25 and 55 and are stored in the home directory of the `cdrexport` user. If the amount of records within the transmission interval exceeds the threshold of 5000 records per file, multiple billing files are created. If no billing records are found for an interval, a billing file without body data is constructed for easy detection of lost billing files on the 3rd party side.

CDR and EDR files are fetched by a 3rd party billing system using SFTP or SCP with either public key or password authentication using the username `cdrexport`.

If public key authentication is chosen, the public key file has to be stored in the file `~/.ssh/authorized_keys2` below the home directory of the `cdrexport` user. Otherwise, a password has to be set for the user.

The 3rd party billing system is responsible for deleting CDR files after fetching them.

---

**Note**

The `cdrexpert` user is kept in a jailed environment on the system, so it has only access to a very limited set of commandline utilities.

---

## 8 Provisioning REST API Interface

The sip:carrier provides the REST API interface for interconnection with 3rd party tools.

The sip:carrier provides a REST API to provision various functionality of the platform. The entry point - and at the same time the official documentation - is at <https://<your-ip>:1443/api>. It allows both administrators and resellers (in a limited scope) to manage the system.

You can either authenticate via username and password of your administrative account you're using to access the admin panel, or via SSL client certificates. Find out more about client certificate authentication in the online API documentation.

### 8.1 API Workflows for Customer and Subscriber Management

The typical tasks done on the API involve managing customers and subscribers. The following chapter focuses on creating, changing and deleting these resources.

The standard life cycle of a customer and subscriber is:

1. Create customer contact
2. Create customer
3. Create subscribers within customer
4. Modify subscribers
5. Modify subscriber preferences (features)
6. Terminate subscriber
7. Terminate customer

The boiler-plate to access the REST API is described in the online API documentation at [/api/#auth](#). A simple example in Perl using password authentication looks as follows:

```
#!/usr/bin/perl -w
use strict;
use v5.10;

use LWP::UserAgent;
use JSON qw();

my $uri = 'https://ngcp.example.com:1443';
my $ua = LWP::UserAgent->new;
my $user = 'myusername';
my $pass = 'mypassword';
$ua->credentials('ngcp.example.com:1443', 'api_admin_http', $user, $pass);
my ($req, $res);
```

For each customer you create, you need to assign a billing profile id. You either have the ID stored somewhere else, or you need to fetch it by searching for the billing profile handle.

```
my $billing_profile_handle = 'my_test_profile';
$req = HTTP::Request->new('GET', "$uri/api/billingprofiles/?handle=$billing_profile_handle" ←
);
$res = $ua->request($req);
if($res->code != 200) {
    die "Failed to fetch billing profile: ".$res->decoded_content."\n";
}
my $billing_profile = JSON::from_json($res->decoded_content);
my $billing_profile_id = $billing_profile->{_embedded}->{'ngcp:billingprofiles'}->{id};
say "Fetched billing profile, id is $billing_profile_id";
```

A customer is mainly a billing container for subscribers without a real identification other than the *external\_id* property you might have stored somewhere else (e.g. the ID of the customer in your CRM). To still easily identify a customer, a customer contact is required. It is created using the */api/customercontacts/* resource.

```
$req = HTTP::Request->new('POST', "$uri/api/customercontacts/");
$req->header('Content-Type' => 'application/json');
$req->content(JSON::to_json({
    firstname => 'John',
    lastname => 'Doe',
    email => 'john.doe@example.com'
}));
$res = $ua->request($req);
if($res->code != 201) {
    die "Failed to create customer contact: ".$res->decoded_content."\n";
}
my $contact_id = $res->header('Location');
$contact_id =~ s/^.+\/(\d+)\$\/$1/; # extract the ID from the Location header
say "Created customer contact, id is $contact_id";
```



### Important

To get the ID of the recently created resource, you need to parse the *Location* header. In future, this approach will be changed for POST requests. The response will also optionally return the ID of the resource. It will be controlled via the *Prefer: return=representation* header as it is already the case for PUT and PATCH.



### Warning

The example above implies the fact that you access the API via a reseller user. If you are accessing the API as the admin user, you also have to provide a *reseller\_id* parameter defining the reseller this contact belongs to.

Once you have created the customer contact, you can create the actual customer.

```
$req = HTTP::Request->new('POST', "$uri/api/customers/");
```

```

$req->header('Content-Type' => 'application/json');
$req->content(JSON::to_json({
    status => 'active',
    contact_id => $contact_id,
    billing_profile_id => $billing_profile_id,
    type => 'sipaccount',
    external_id => undef, # can be set to your crm's customer id
}));
$res = $ua->request($req);
if($res->code != 201) {
    die "Failed to create customer: ".$res->decoded_content."\n";
}
my $customer_id = $res->header('Location');
$customer_id =~ s/^.+\/(\d+)/$1/; # extract the ID from the Location header
say "Created customer, id is $customer_id";

```

Once you have created the customer, you can add subscribers to it. One customer can hold multiple subscribers, up to the *max\_subscribers* property which can be set via */api/customers/*. If this property is not defined, a virtually unlimited number of subscribers can be added.

```

$req = HTTP::Request->new('POST', "$uri/api/subscribers/");
$req->header('Content-Type' => 'application/json');
$req->content(JSON::to_json({
    status => 'active',
    customer_id => $customer_id,
    primary_number => { cc => 43, ac => 9876, sn => 10001 }, # the main number
    alias_numbers => [ # as many alias numbers the subscriber can be reached at (or skip ←
        param if none)
        { cc => 43, ac => 9877, sn => 10001 },
        { cc => 43, ac => 9878, sn => 10001 }
    ],
    username => 'test_10001',
    domain => 'ngcp.example.com',
    password => 'secret subscriber pass',
    webusername => 'test_10001',
    webpassword => undef, # set undef if subscriber shouldn't be able to log into sipwise ←
        csc
    external_id => undef, # can be set to the operator crm's subscriber id
}));
$res = $ua->request($req);
if($res->code != 201) {
    die "Failed to create subscriber: ".$res->decoded_content."\n";
}
my $subscriber_id = $res->header('Location');
$subscriber_id =~ s/^.+\/(\d+)/$1/; # extract the ID from the Location header
say "Created subscriber, id is $subscriber_id";

```

**Important**

A domain must exist before creating a subscriber. You can create the domain via `/api/domains/`.

At that stage, the subscriber can connect both via SIP and XMPP, and can be reached via the primary number, all alias numbers, as well as via the SIP URI.

If you want to set call forwards for the subscribers, then perform an API call as follows.

```
$req = HTTP::Request->new('PUT', "$uri/api/callforwards/$subscriber_id");
$req->header('Content-Type' => 'application/json');
$req->header('Prefer' => "return=minimal"); # use return=representation to get full json ←
      response
$req->content(JSON::to_json({
  cfna => { # set a call-forward if subscriber is not registered
    destinations => [
      { destination => "4366610001", timeout => 10 }, # ring this for 10s
      { destination => "4366710001", timeout => 300}, # if no answer, ring that for ←
        300s
    ],
    times => undef # no time-based call-forward, trigger cfna always
  }
}));
$res = $ua->request($req);
if($res->code != 204) { # if return=representation, it's 200
  die "Failed to set cfna for subscriber: ".$res->decoded_content."\n";
}
```

You can set cfu, cfna, cft and cft via this API call, also all at once. Destinations can be hunting lists as described above or just a single number. Also, a time set can be provided to trigger call forwards only during specific time periods.

To provision certain features of a subscriber, you can manipulate the subscriber preferences. You can find a full list of preferences available for a subscriber at `/api/subscriberpreferencedefs/`.

```
$req = HTTP::Request->new('GET', "$uri/api/subscriberpreferences/$subscriber_id");
$res = $ua->request($req);
if($res->code != 200) {
  die "Failed to fetch subscriber preferences: ".$res->decoded_content."\n";
}
my $prefs = JSON::from_json($res->decoded_content);
delete $prefs->{__links}; # not needed in update

$prefs->{prepaid_library} = 'libinewrate'; # switch to inew billing
$prefs->{block_in_clir} = JSON::true; # reject incoming anonymous calls
$prefs->{block_in_list} = [ # reject calls from the following numbers:
  '4366412345', # this particular number
  '431*', # all vienna/austria numbers
```

```

];
$req = HTTP::Request->new('PUT', "$uri/api/subscriberpreferences/$subscriber_id");
$req->header('Content-Type' => 'application/json');
$req->header('Prefer' => "return=minimal"); # use return=representation to get full json ←
      response
$req->content(JSON::to_json($prefs));
$res = $ua->request($req);
if($res->code != 204) {
    die "Failed to update subscriber preferences: ".$res->decoded_content."\n";
}
say "Updated subscriber preferences";

```

Modifying numbers assigned to a subscriber, changing the password, locking a subscriber, etc. can be done directly on the subscriber resource.

```

$req = HTTP::Request->new('GET', "$uri/api/subscribers/$subscriber_id");
$res = $ua->request($req);
if($res->code != 200) {
    die "Failed to fetch subscriber: ".$res->decoded_content."\n";
}
my $sub = JSON::from_json($res->decoded_content);
delete $sub->{_links}; # not needed in update
push @{$sub->{alias_numbers}}, { cc => 1, ac => 5432, sn => $t }; # add this number
push @{$sub->{alias_numbers}}, { cc => 1, ac => 5433, sn => $t }; # add another number

$req = HTTP::Request->new('PUT', "$uri/api/subscribers/$subscriber_id");
$req->header('Content-Type' => 'application/json');
$req->header('Prefer' => "return=minimal"); # use return=representation to get full json ←
      response
$req->content(JSON::to_json($sub));
$res = $ua->request($req);
if($res->code != 204) {
    die "Failed to update subscriber: ".$res->decoded_content."\n";
}
say "Updated subscriber";

```

At the end of a subscriber life cycle, it can be terminated. Once terminated, you can NOT recover the subscriber anymore.

```

$req = HTTP::Request->new('DELETE', "$uri/api/subscribers/$subscriber_id");
$res = $ua->request($req);
if($res->code != 204) {
    die "Failed to terminate subscriber: ".$res->decoded_content."\n";
}
say "Terminated subscriber";

```

Note that certain information is still available in the internal database to perform billing/rating of calls done by this subscriber. Nevertheless, the data is removed from the operational tables of the database, so the subscriber is not able to connect to the system, login or make calls/chats.



Resources modification can be done via the GET/PUT combination. Alternatively, you can add, modify or delete single properties of a resource without actually fetching the whole resource. See an example below where we terminate the status of a customer using the PATCH method.

```
$req = HTTP::Request->new('PATCH', "$uri/api/customers/$customer_id");
$req->header('Content-Type' => 'application/json-patch+json');
$req->header('Prefer' => "return=minimal"); # use return=representation to get full json ↔
      response
$req->content(JSON::to_json([
  { op => 'replace', path => '/status', value => 'terminated' }
]));
$res = $ua->request($req); # this will also terminate all still active subscribers
if($res->code != 204) {
  die "Failed to terminate customer: ".$res->decoded_content."\n";
}
say "Terminated customer";
```

## 8.2 API performance considerations

The REST API is designed with pagination support built-in. It is mandatory, to implement pagination in your API clients. If you circumvent pagination by setting the number of rows requested in one API call to a very high number the following side effects may appear:

1. An HTTP timeout at the gateway may appear. The default timeout limit is set to 60s. It can be modified via a customtt file: */etc/ngcp-config/templates/etc/nginx/sites-available/ngcp-panel\_admin\_api.customtt.tt2*.
2. Other parts of the system may become unresponsive due to mysql table locks. This especially applies to endpoints related to the Customers entity.

## 9 Configuration Framework

The sip:carrier provides a configuration framework for consistent and easy to use low level settings management. A basic usage of the configuration framework only needs two actions already used in previous chapters:

- Edit `/etc/ngcp-config/config.yml` file.
- Execute `ngcpcfg apply 'my commit message'` command.

Low level management of the configuration framework might be required by advanced users though. This chapter explains the architecture and usage of the NGCP configuration framework. If the basic usage explained above fits your needs, feel free to skip this chapter and return to it when your requirements change.

A more detailed workflow of the configuration framework for creating a configuration file consists of 7 steps:

- Generation or editing of configuration templates and/or configuration values.
- Generation of the configuration files based on configuration templates and configuration values defined in `config.yml`, `constants.yml` and `network.yml` files.
- Execution of `prebuild` commands if defined for a particular configuration file or configuration directory.
- Placement of the generated configuration file in the target directory. This step is called `build` in the configuration framework.
- Execution of `postbuild` commands if defined for that configuration file or configuration directory.
- Execution of `services` commands if defined for that configuration file or configuration directory. This step is called `services` in the configuration framework.
- Saving of the generated changes. This step is called `commit` in the configuration framework.

### 9.1 Configuration templates

The sip:carrier provides configuration file templates for most of the services it runs. These templates are stored in the directory `/etc/ngcp-config/templates`.

Example: Template files for `/etc/ngcp-sems/sems.conf` are stored in `/etc/ngcp-config/templates/etc/ngcp-sems/`.

There are different types of files in this template framework, which are described below.

#### 9.1.1 .tt2 and .customtt.tt2 files

These files are the main template files that will be used to generate the final configuration file for the running service. They contain all the configuration options needed for a running sip:carrier system. The configuration framework will combine these files with the values provided by `config.yml`, `constants.yml` and `network.yml` to generate the appropriate configuration file.

Example: Let's say we are changing the IP used by kamailio load balancer on interface `eth0` to IP 1.2.3.4. This will change kamailio's listen IP address, when the configuration file is generated. A quick look to the template file under `/etc/ngcp-config/templates/etc/kamailio/` will show a line like this:

```
listen=udp:[% ip %]:[% kamailio.lb.port %]
```

After applying the changes with the `ngcpconf apply 'my commit message'` command, a new configuration file will be created under `/etc/kamailio/lb/kamailio.cfg` with the proper values taken from the main configuration files (in this case `network.yml`):

```
listen=udp:1.2.3.4:5060
```

All the low-level configuration is provided by these `.tt2` template files and the corresponding `config.yml` file. Anyway, advanced users might require a more particular configuration.

Instead of editing `.tt2` files, the configuration framework recognises `.customtt.tt2` files. These files are the same as `.tt2`, but they have higher priority when the configuration framework creates the final configuration files. An advanced user should create a `.customtt.tt2` file from a copy of the corresponding `.tt2` template and leave the `.tt2` template untouched. This way, the user will have his personalized configuration and the system will continue providing a working, updated configuration template in `.tt2` format.

Example: We'll create `/etc/ngcp-config/templates/etc/lb/kamailio.cfg.customtt.tt2` and use it for our personalized configuration. In this example, we'll just append a comment at the end of the template.

```
cd /etc/ngcp-config/templates/etc/kamailio/lb
cp kamailio.cfg.tt2 kamailio.cfg.customtt.tt2
echo '# This is my last line comment' >> kamailio.cfg.customtt.tt2
ngcpconf apply 'my commit message'
```

The `ngcpconf` command will generate `/etc/kamailio/kamailio.cfg` from our custom template instead of the general one.

```
tail -1 /etc/kamailio/kamailio.cfg
# This is my last line comment
```

---

### Tip

The `tt2` files use the [Template Toolkit](#) language. Therefore you can use all the feature this excellent toolkit provides within `ngcpconf`'s template files (all the ones with the `.tt2` suffix).

---

## 9.1.2 .prebuild and .postbuild files

After creating the configuration files, the configuration framework can execute some commands before and after placing that file in its target directory. These commands usually are used for changing the file's owner, groups, or any other attributes. There are some rules these commands need to match:

- They have to be placed in a `.prebuild` or `.postbuild` file in the same path as the original `.tt2` file.
- The file name must be the same as the configuration file, but having the mentioned suffixes.
- The commands must be `bash` compatible.
- The commands must return 0 if successful.

- The target configuration file is matched by the environment variable `output_file`.

Example: We need `www-data` as owner of the configuration file `/etc/ngcp-ossbss/provisioning.conf`. The configuration framework will by default create the configuration files with `root:root` as owner:group and with the same permissions (`rwX`) as the original template. For this particular example, we will change the owner of the generated file using the `.postbuild` mechanism.

```
echo 'chgrp www-data ${output_file}' \
> /etc/ngcp-config/templates/etc/ngcp-ossbss/provisioning.conf.postbuild
```

### 9.1.3 .services files

`.services` files are pretty similar and might contain commands that will be executed after the `build` process. There are two types of `.services` files:

- The particular one, with the same name as the configuration file it is associated to.  
Example: `/etc/ngcp-config/templates/etc/asterisk/sip.conf.services` is associated to `/etc/asterisk/sip.conf`
- The general one, named `ngcpcfg.services` which is associated to every file in its target directory.  
Example: `/etc/ngcp-config/templates/etc/asterisk/ngcpcfg.services` is associated to every file under `/etc/asterisk/`

When the `services` step is triggered all `.services` files associated to a changed configuration file will be executed. In case of the general file, any change to any of the configuration files in the directory will trigger the execution of the commands.

---

#### Tip

If the service script has the execute flags set (`chmod +x $file`) it will be invoked directly. If it doesn't have execute flags set it will be invoked under `bash`. Make sure the script is `bash` compatible if you do not set execute permissions on the service file.

---

These commands are usually service reload/restarts to ensure the new configuration has been loaded by running services.

---

#### Note

The configuration files mentioned in the following example usually already exist on the platform. Please make sure you don't overwrite any existing files if following this example.

---

Example:

```
echo '/etc/init.d/mysql restart' \
> /etc/ngcpcfg-config/templates/etc/mysql/my.cnf.services
echo '/etc/init.d/asterisk restart' \
> /etc/ngcpcfg-config/templates/etc/asterisk/ngcpcfg.services
```

In this example we created two `.services` files. Now, each time we trigger a change to `/etc/mysql/my.cnf` or to `/etc/asterisk/*` we'll see that MySQL or Asterisk services will be restarted by the `ngcpcfg` system.

## 9.2 config.yml, constants.yml and network.yml files

The `/etc/ngcp-config/config.yml` file contains all the user-configurable options, using the **YAML** (YAML Ain't Markup Language) syntax.

The `/etc/ngcp-config/constants.yml` file provides configuration options for the platform that aren't supposed to be edited by the user. Do not manually edit this file unless you really know what you're doing.

The `/etc/ngcp-config/network.yml` file provides configuration options for all interfaces and IP addresses on those interfaces. You can use the `ngcp-network` tool for conveniently change settings without having to manually edit this file.

The `/etc/ngcp-config/ngcpcfg.cfg` file is the main configuration file for `ngcpcfg` itself. Do not manually edit this file unless you really know what you're doing.

## 9.3 ngcpcfg and its command line options

The shared storage used by all nodes is the shared storage of the mgmt pair.

The `ngcpcfg` utility supports the following command line options:

### 9.3.1 apply

The `apply` option is a short-cut for the options "check && build && services && commit" and also executes `etckeeper` to record any modified files inside `/etc`. It is the recommended option to use the `ngcpcfg` framework unless you want to execute any specific commands as documented below.

### 9.3.2 build

The `build` option generates (and therefore also updates) configuration files based on their configuration (`config.yml`) and template files (`.tt2`). Before the configuration file is generated a present `.prebuild` will be executed, after generation of the configuration file the according `.postbuild` script (if present) will be executed. If a `file` or `directory` is specified as argument the build will generate only the specified configuration file/directory instead of running through all present templates.

Example: to generate only the file `/etc/nginx/sites-available/ngcp-panel` you can execute:

```
ngcpcfg build /etc/nginx/sites-available/ngcp-panel
```

Example: to generate all the files located inside the directory `/etc/nginx/` you can execute:

```
ngcpcfg build /etc/nginx/
```

### 9.3.3 commit

The `commit` option records any changes done to the configuration tree inside `/etc/ngcp-config`. The `commit` option should be executed when you've modified anything inside the configuration tree.

### 9.3.4 decrypt

Decrypt `/etc/ngcp-config-encrypted.tgz.gpg` and restore configuration files, doing the reverse operation of the *encrypt* option. Note: This feature is only available if the `ngcp-ngcpcfg-locker` package is installed.

### 9.3.5 diff

Show uncommitted changes between `ngcpcfg`'s Git repository and the working tree inside `/etc/ngcp-config`. If the tool doesn't report anything it means that there are no uncommitted changes. If the `--addremove` option is specified then new and removed files (iff present) that are not yet (un)registered to the repository will be reported, no further diff actions will be executed then. Note: This option is available since `ngcp-ngcpcfg` version 0.11.0.

### 9.3.6 encrypt

Encrypt `/etc/ngcp-config` and all resulting configuration files with a user defined password and save the result as `/etc/ngcp-config-encrypted.tgz.gpg`. Note: This feature is only available if the `ngcp-ngcpcfg-locker` package is installed.

### 9.3.7 help

The *help* options displays `ngcpcfg`'s help screen and then exits without any further actions.

### 9.3.8 initialise

The *initialise* option sets up the `ngcpcfg` framework. This option is automatically executed by the installer for you, so you shouldn't have to use this option in normal operations mode.

### 9.3.9 pull

Retrieve modifications from shared storage. Note: This option is available in the High Availability setup only.

### 9.3.10 push

Push modifications to shared storage and remote systems. After changes have been pushed to the nodes the *build* option will be executed on each remote system to rebuild the configuration files (unless the `--nobuild` has been specified, then the build step will be skipped). If hostname(s) or IP address(es) is given as argument then the changes will be pushed to the shared storage and to the given hosts only. You can use *all* as a shortcut to push to the other nodes. If no host has been specified then the hosts specified in `/etc/ngcp-config/systems.cfg` are used. Note: This option is available in the High Availability setup only.

### 9.3.11 services

The *services* option executes the service handlers for any modified configuration file(s)/directory.

### 9.3.12 status

The *status* option provides a human readable interface to check the state of the configuration tree. If you are unsure what should be done as next step or if want to check the current state of the configuration tree just invoke *ngcpcfg status*.

If everything is OK and nothing needs to be done the output should look like:

```
# ngcpcfg status
Checking state of ngcpcfg:
OK:  has been initialised already (without shared storage)
Checking state of configuration files:
OK:  nothing to commit.
Checking state of /etc files
OK:  nothing to commit.
```

If the output doesn't say "OK" just follow the instructions provided by the output of *ngcpcfg status*.

Further details regarding the *ngcpcfg* tool are available through *man ngcpcfg* on the Sipwise Next Generation Platform.

## 10 Network Configuration

Starting with version 2.7, the sip:carrier uses a dedicated *network.yml* file to configure the IP addresses of the system. The reason for this is to be able to access all IPs of all nodes for all services from any particular node in case of a distributed system on one hand, and in order to be able to generate */etc/network/interfaces* automatically for all nodes based on this central configuration file.

### 10.1 General Structure

The basic structure of the file looks like this:

```
hosts:
  self:
    role:
      - proxy
      - lb
      - mgmt
    interfaces:
      - eth0
      - lo
    eth0:
      ip: 192.168.51.213
      netmask: 255.255.255.0
      type:
        - sip_ext
        - rtp_ext
        - web_ext
        - web_int
    lo:
      ip: 127.0.0.1
      netmask: 255.255.255.0
      type:
        - sip_int
        - ha_int
```

Some more complete, sample configuration is shown in [network.yml Overview](#) Section B.3 section of the handbook.

The file contains all configuration parameters under the main key: `hosts`

In sip:carrier systems all hosts of the system are defined, and the names are the actual host names instead of *self*, like this:

```
hosts:

  web01a:
    peer: web01b
    role: ...
    interfaces: ...
```



```
web01b:
  peer: web01a
  role: ...
  interfaces: ...
```

### 10.1.1 Available Host Options

There are three different main sections for a host in the config file, which are *role*, *interfaces* and the actual interface definitions.

- *role*: The role setting is an array defining which logical roles a node will act as. Possible entries for this setting are:
  - *mgmt*: This entry means the host is acting as management node for the platform. In a sip:carrier system this option must always be set. The management node exposes the admin and CSC panels to the users and the APIs to external applications and is used to export CDRs. Please note: this is only set on the nodes of the management pairs. This node is also the source of the installations of other nodes via iPXE and has the *approx* service (apt proxy).
  - *lb*: This entry means the host is acting as SIP load-balancer for the platform. In a sip:carrier system this option must always be set. Please note: this is only set on the nodes of the *lb* pairs. The SIP load-balancer acts as an ingress and egress point for all SIP traffic to and from the platform.
  - *proxy*: This entry means the host is acting as SIP proxy for the platform. In a sip:carrier system this option must always be set. Please note: this is only set on the nodes of the *proxy* pairs. The SIP proxy acts as registrar, proxy and application server and media relay, and is responsible for providing the features for all subscribers provisioned on it.
  - *db*: This entry means the host is acting as the database node for the platform. In a sip:carrier system this option must always be set. Please note: this is only set on the nodes of the *db* pairs. The database node exposes the MySQL and Redis databases.
  - *rtp*: This entry means the host is acting as the RTP relay node for the platform. In a sip:carrier system this option must always be set. Please note: this is only set on the nodes of the *RTP relay* pairs. The RTP relay node runs the *rtpengine* NGCP component.
  - *li*: This entry means the host is acting as the interface towards a lawful interception service provider.
- *interfaces*: The interfaces setting is an array defining all interface names in the system. The actual interface details are set in the actual interface settings below. It typically includes `lo`, `eth0`, `eth1` physical and a number of virtual interfaces, like: `bond0`, `vlanXXX`
- *<interface name>*: After the interfaces are defined in the *interfaces* setting, each of those interfaces needs to be specified as a separate set of parameters.

Additional main parameters of a node:

- *dbnode*: the sequence number (unique ID) of the node in the database cluster; the value is used only if main DB is set up as an extended cluster on other than *db0x* nodes too
- *peer*: the hostname of the peer node within the pair of nodes (e.g. "web01b" for *web01a* host). The purpose of that: each node knows its companion for providing high availability, data replication etc.

- *status*: one of *online*, *offline*, *inactive*. *inactive* means that the node is up but is not ready to work in the cluster (installing process). *offline* means that the node is not reachable. *online* is a normal working node.

### 10.1.2 Interface Parameters

- *hwaddr*: MAC address of the interface



#### Caution

This *must* be filled in properly for the interface that is used as type *ha\_int*, because the value of it will be used during the boot process of the installation of nodes via iPXE, if PXE-boot is enabled.

---

- *ip*: IPv4 address of the node
- *v6ip*: IPv6 address of the node; optional
- *netmask*: IPv4 netmask
- *shared\_ip*: shared IPv4 address of the pair of nodes; this is a list of addresses
- *shared\_v6ip*: shared IPv6 address of the pair of nodes; optional; this is a list of addresses
- *advertised\_ip*: the IP address that is used in SIP messages when the NGCP system is behind NAT/SBC. An example of such a deployment is *Amazon AMI*, where the server doesn't have a public IP, so *load-balancer* component of NGCP needs to know what his public domain is (→ *advertised\_ip*).
- *type*: type of services that the node provides; these are usually the VLANs defined for a particular NGCP system.

---

#### Note

You can assign a type only once per node.

---

Available types are:

- *api\_int*: internal, API-based communication interface. It is used for the internal communication of such services as faxserver, fraud detection and others.
- *aux\_ext*: interface for potentially insecure external components like remote system log collection service.

---

#### Note

For example the *CloudPBX* module can use it to provide time services and remote logging facilities to end customer devices. The type *aux\_ext* is assigned to *lo* interface by default. If it is needed to expose this type to the public, it is recommended to assign the type *aux\_ext* to a separate VLAN interface to be able to limit or even block the incoming traffic easily via firewalling in case of emergency, like a (D)DoS attack on external services.

---

- *mon\_ext*: remote monitoring interface (e.g. SNMP)
  - *rtp\_ext*: main (external) interface for media traffic
  - *sip\_ext*: main (external) interface for SIP signalling traffic between NGCP and other SIP endpoints
-

- `sip_ext_incoming`: additional, optional interface for incoming SIP signalling traffic
- `sip_int`: internal SIP interface used by NGCP components (*lb, proxy, etc.*)
- `ssh_ext`: command line (SSH) remote access interface
- `web_ext`: interface for web-based or API-based provisioning and administration
- `web_int`: interface for the administrator's web panel, his API and generic internal API communication
- `li_int`: used for LI (Lawful Interception) traffic routing
- `ha_int`: main communication interface between the nodes
- `boot_int`: the default VLAN used to install nodes via PXE-boot method
- `rtp_int`: internal interface for handling RTP traffic among NGCP nodes that may reside in greater distance from each other, like in case of a specialised NGCP configuration with centralized web / DB / proxy nodes and distributed LB nodes (Please refer to [Cluster Sets](#) Section 10.2.5 section for further details)

---

### Note

Please note that, apart from the standard ones described so far, there might be other *types* defined for a particular NGCP system.

---

- `vlan_raw_device`: tells which physical interface is used by the particular VLAN
- `post_up`: routes can be defined here (interface-based routing)
- `bond_XY`: specific to "bond0" interface only; these contain Ethernet bonding properties

## 10.2 Advanced Network Configuration

You have a typical deployment now and you are good to go, however you may need to do extra configuration depending on the devices you are using and functionality you want to achieve.

### 10.2.1 Extra SIP Sockets

By default, the load-balancer listens on the UDP and TCP ports 5060 (*kamailio*→*lb*→*port*) and TLS port 5061 (*kamailio*→*lb*→*tls*→*port*). If you need to setup one or more extra SIP listening ports or IP addresses in addition to those standard ports, please edit the *kamailio*→*lb*→*extra\_sockets* option in your `/etc/ngcp-config/config.yml` file.

The correct format consists of a label and value like this:

```
extra_sockets:
  port_5064: udp:10.15.20.108:5064
  test: udp:10.15.20.108:6060
```

The label is shown in the `outbound_socket` peer preference (if you want to route calls to the specific peer out via specific socket); the value must contain a transport specification as in example above (udp, tcp or tls). After adding execute `ngcpcfg apply`:

```
ngcpcfg apply 'added extra socket' && ngcpcfg push all
```

The direction of communication through this SIP extra socket is incoming+outgoing. The sip:carrier will answer the incoming client registrations and other methods sent to the extra socket. For such incoming communication no configuration is needed. For the outgoing communication the new socket must be selected in the `outbound_socket` peer preference. For more details read the next section Section 10.2.2 that covers peer configuration for SIP and RTP in greater detail.



### Important

In this section you have just added an extra SIP socket. RTP traffic will still use your `rtp_ext` IP address.

## 10.2.2 Extra SIP and RTP Sockets

If you want to use an additional interface (with a different IP address) for SIP signalling and RTP traffic you need to add your new interface in the `/etc/network/interfaces` file. Also the interface must be declared in `/etc/ngcp-config/network.yml`.

Suppose we need to add a new SIP socket and a new RTP socket on VLAN 100. You can use the `ngcp-network` tool for adding interfaces without having to manually edit this file:

```
ngcp-network --set-interface=eth0.100 --host=slb01a --ip=auto --netmask=auto --hwaddr=auto <-
  --type=sip_ext_incoming
ngcp-network --set-interface=eth0.100 --host=slb01b --ip=auto --netmask=auto --hwaddr=auto <-
  --type=sip_ext_incoming
ngcp-network --set-interface=eth0.100 --host=prx01a --ip=auto --netmask=auto --hwaddr=auto <-
  --type=rtp_int_100
ngcp-network --set-interface=eth0.100 --host=prx01b --ip=auto --netmask=auto --hwaddr=auto <-
  --type=rtp_int_100
```

The generated file should look like the following:

```
slb01a:
..
..
  eth0.100:
    hwaddr: ff:ff:ff:ff:ff:ff
    ip: 192.168.1.2
    netmask: 255.255.255.0
    shared_ip:
      - 192.168.1.3
    shared_v6ip: ~
    type:
      - sip_ext_incoming
..
..
  interfaces:
```

```
- lo
- eth0
- eth0.100
- eth1
..
..
prx01a:
..
..
  eth0.100:
    hwaddr: ff:ff:ff:ff:ff:ff
    ip: 192.168.1.20
    netmask: 255.255.255.0
    shared_ip:
      - 192.168.1.30
    shared_v6ip: ~
    type:
      - rtp_int_100
..
..
  interfaces:
    - lo
    - eth0
    - eth0.100
    - eth1
..
..
slb01b:
..
..
  eth0.100:
    hwaddr: ff:ff:ff:ff:ff:ff
    ip: 192.168.1.4
    netmask: 255.255.255.0
    shared_ip:
      - 192.168.1.3
    shared_v6ip: ~
    type:
      - sip_ext_incoming
..
..
  interfaces:
    - lo
    - eth0
    - eth0.100
    - eth1
..
..
```

```

prx01b:
..
..
  eth0.100:
    hwaddr: ff:ff:ff:ff:ff:ff
    ip: 192.168.1.40
    netmask: 255.255.255.0
    shared_ip:
      - 192.168.1.30
    shared_v6ip: ~
    type:
      - rtp_int_100
..
..
  interfaces:
    - lo
    - eth0
    - eth0.100
    - eth1

```

As you can see from the above example, extra SIP interfaces must have type *sip\_ext\_incoming*. While *sip\_ext* should be listed only once per host, there can be multiple *sip\_ext\_incoming* interfaces. The direction of communication through this SIP interface is incoming only. The sip:carrier will answer the incoming client registrations and other methods sent to this address and remember the interfaces used for clients' registrations to be able to send incoming calls to him from the same interface.

In order to use the interface for the outbound SIP communication it is necessary to add it to extra\_sockets section in */etc/ngcp-config/config.yml* and select in the *outbound\_socket* peer preference. So if using the above example we want to use the vlan100 IP as source interface towards a peer, the corresponding section may look like the following:

```

extra_sockets:
  port_5064: udp:10.15.20.108:5064
  test: udp:10.15.20.108:6060
  int_100: udp:192.168.1.3:5060

```

The changes have to be applied:

```
ngcpcfg apply 'added extra SIP and RTP socket' && ngcpcfg push all
```

After applying the changes, a new SIP socket will listen on IP 192.168.1.3 on slb01 node and this socket can now be used as source socket to send SIP messages to your peer for example. In above example we used label *int\_100*. So the new label "int\_100" is now shown in the *outbound\_socket* peer preference.

Also, RTP socket is now listening on 192.168.1.30 on prx01 node and you can choose the new RTP socket to use by setting parameter *rtp\_interface* to the Label "int\_100" in your Domain/Subscriber/Peer preferences.

### 10.2.3 Alternative RTP Interface Selection Using ICE

Normally, each interface that was configured with a type that starts with `rtp_` can be selected individually as RTP interface in the Domain/Subscriber/Peer preferences. For example, if the interface types `rtp_ext`, `rtp_int`, and `rtp_int_100` have been configured, the Domain/Subscriber/Peer preferences will allow the RTP interfaces to be selected as either `ext`, `int`, or `int_100` in addition to "default".

The same `rtp_` interface type can be configured on multiple interfaces. If this is the case, and if ICE (*Interactive Connectivity Establishment*) is enabled for a Domain/Subscriber/Peer, it is possible to use ICE to automatically negotiate which interface should be used for RTP communications. ICE must be supported by the remote client for this to work.

For example, `rtp_ext` can be configured on multiple interfaces like so (abbreviated):

```
..
..
  eth0.100:
    type:
      - rtp_ext
..
  eth0.150:
    type:
      - rtp_ext
..
  eth1:
    type:
      - rtp_ext
..
..
```

In this example, the RTP interface `ext` will be available for selection in the Domain/Subscriber/Peer preferences. If selected and if ICE is enabled, the addresses of all three interfaces will be presented to the remote client, and ICE will be used to negotiate which one of them will be used for communications. This can be useful in multi-homed environments, or when remote clients are on private networks.

### 10.2.4 Extended RTP Port Range Using Multiple Interfaces

If the RTP port range configured via the `config.yml` keys `rtpproxy.minport` and `rtpproxy.maxport` is not sufficient to handle all concurrent calls, it is possible to load-balance the RTP ports across multiple interfaces. This is useful if the RTP proxy runs out of ports and if not enough additional ports are available.

To enable this, multiple interfaces with different addresses must be configured, and interface types of the format `rtp_NAME:SUFFIX` must be assigned to them. For example, if the RTP interface named `ext` should be load-balanced across three interfaces, they can be configured like so (abbreviated):

```
..
..
  eth0.100:
    type:
```

```

    - rtp_ext:1
..
  eth0.150:
    type:
      - rtp_ext:2
..
  eth1:
    type:
      - rtp_ext:3
..
..

```

In this example, all three given RTP interface types will be available for selection in the Domain/Subscriber/Peer preferences individually (as *ext:1* and so on), but in addition to that, an interface named just *ext* will also be available for selection. If *ext* is selected, only one of the three RTP interfaces will be selected in a round-robin fashion, thus increasing the number of available RTP ports threefold. The round-robin algorithm only selects an interface if it actually has RTP ports available.

### 10.2.5 Cluster Sets

In a sip:carrier system it is possible to have geographically distributed nodes in the same logical NGCP unit. Such a configuration typically involves the following elements:

- **centralised** management (*web*), database (*db*) and proxy (*prx*) nodes: these provide all higher level functionality, like system administration, subscriber registration, call routing, etc.
- **distributed** load balancer (*lb*) nodes: these serve as SBCs for the the whole NGCP and handle SIP and RTP traffic to / from SIP endpoints (e.g. subscribers); and they also communicate with the central elements of NGCP (e.g. proxy nodes)

In case of such an NGCP node configuration it is possible to define *cluster sets* which are collections of NGCP nodes providing the load balancer functionality.

Cluster sets can be assigned to subscriber *domains* or *SIP peers* and will determine the route of SIP and RTP traffic for those sets of SIP endpoints:

- For *SIP peers* the selected nodes will be used to send outbound SIP traffic through
- For both *SIP peers* and subscriber *domains* the selected nodes will provide RTP relay functionality (the *rtpengine* NGCP component will run on those nodes)

#### 10.2.5.1 Configuration of Nodes of Cluster Sets

There are 2 places in NGCP's main configuration files where an entry for cluster sets must be inserted:

1. Declaration of cluster sets

This happens in `/etc/ngcp-config/config.yml` file, see an example below:



```
cluster_sets:
  default:
    dispatcher_id: 50
  default_set: default
  poland:
    dispatcher_id: 51
  type: distributed
```

Configuration entries are:

- `<label>`: an arbitrary label of the cluster set; in the above example we have 2 of them: `default` and `poland`; the cluster set `default` is always defined, even if cluster sets are not used
- `<label>.dispatcher_id`: a unique, numeric value that identifies a particular cluster set
- `default_set`: selects the default cluster set
- `type`: the type of cluster set; can be `central` or `distributed`

## 2. Assignment of cluster sets

This happens in `/etc/ngcp-config/network.yml` file, see an example below:

```
.
.
lb03a:
  .
  .
  vlan792:
    cluster_sets:
      - poland
    hwaddr: 00:00:00:00:00:00
    ip: 172.30.61.37
    netmask: 255.255.255.240
    shared_ip: 172.30.61.36
    type:
      - sip_int
    vlan_raw_device: bond0
```

In the network configuration file typically the load balancer (*lb*) nodes are assigned to cluster sets. More precisely: network interfaces of load balancer nodes that have `sip_int` type—that are used for SIP signalling and NGCP's internal *rtengine* command protocol—are assigned to cluster sets.

In order to do such an assignment a cluster set's label has to be added to the `cluster_sets` parameter, which is a list.

After modifying network configuration with cluster sets, the new configuration must be applied in the usual way:

```
> ngcpcfg apply 'Added cluster sets'
> ngcpcfg push all
```

### 10.2.5.2 Configuration of Cluster Sets for SIP and RTP Traffic

For both SIP peers and subscriber domains you can select the cluster set labels predefined in `config.yml` file.

- **SIP peers:** In order to select a particular cluster set for a SIP peer you have to navigate to *Peerings* → *select the peering group* → *select the peering server* → *Preferences* → *NAT and Media Flow Control* and then *Edit* `lbrtp_set` parameter.

**Peer Host "Vlada01" - Preferences**

← Back
★ Flash Dialogic

Expand Groups

Access Restrictions

Number Manipulations

NAT and Media Flow Control

	Attribute	Name	Value	
ⓘ	use_rtproxy	RTP-Proxy Mode	Always with plain SDP	
ⓘ	ipv46_for_rtproxy	IPv4/IPv6 bridging mode	Auto-detect	
ⓘ	lbrtp_set	The cluster set used for SIP lb and RTP	None	<span style="border: 2px solid red; border-radius: 50%; padding: 2px 5px; color: white; font-size: 0.8em;">Edit</span>
ⓘ	rtp_interface	RTP interface	default	

Figure 96: Select Cluster Set for a Peer

- **Domains:** In order to select a particular cluster set for a domain you have to navigate to *Domains* → *select the domain* → *Preferences* → *NAT and Media Flow Control* and then *Edit* `lbrtp_set` parameter.

## Domain "195.185.37.60" - Preferences

← Back

Expand Groups

Call Blockings

Access Restrictions

Number Manipulations

NAT and Media Flow Control

	Attribute	Name	Value	
ⓘ	sound_set	System Sound Set	<input type="text"/>	
ⓘ	no_nat_sipping	Disable NAT SIP pings	<input type="checkbox"/>	
ⓘ	use_rtpproxy	RTP-Proxy Mode	Always with plain SDP	
ⓘ	ipv46_for_rtpproxy	IPv4/IPv6 bridging mode	Auto-detect	
ⓘ	bypass_rtpproxy	Disable RTP-Proxy in the selected case	Never	
ⓘ	lbrtp_set	The cluster set used for SIP lb and RTP	None	<input type="button" value="Edit"/>
ⓘ	rtp_interface	RTP interface	default	

Figure 97: Select Cluster Set for a Domain

## 11 Licenses

The Sipwise NGCP—starting from mr5.5.1 release—implements *software licensing* in form of a regular comparison of the licensed services and capacities against the actual usage patterns of the platform. The purpose of this function is to monitor system usage and to raise warnings to the platform operator if the thresholds of commercially agreed license parameters (like number of provisioned subscribers or number of concurrent calls) are exceeded.

### 11.1 What is Subject to Licensing?

Sipwise NGCP licenses determine 2 groups of system parameters which are regularly compared with actual values gathered from the system:

- **performance parameters:**
  - number of provisioned subscribers
  - number of registered subscribers
  - number of concurrent calls
- **feature parameters:** additional features / services that are subject to commercial agreement:
  - pre-paid billing
  - CPBX (Cloud PBX) services
  - Push notifications (mobile SIP clients on iOS and Android)
  - Lawful Interception services
  - Call history available on web interface of NGCP

### 11.2 How Licensing Works

Sipwise operates a *licensing server* that is the source of license data for each deployed NGCP node. The nodes themselves request licensing data from the license server regularly and compare them with actual system performance indicators, check the activated features against the licensed ones. The presence and activity of the *license client* module ("licensed" process) may be confirmed by checking e.g. the output of "monit summary" command. It should contain a line showing:

```
licensed    Running
```

All nodes of a single NGCP installation share the same license key. This is also valid for geographically distributed setups. This license key is referred by an ID that has to be configured in the main NGCP configuration file (config.yml), and that ID will be used to request license data from the license server.

In order for the license validation to work each node of an NGCP installation must be able to connect to the Sipwise license server via standard HTTPS protocol (TCP, port 443). Alternatively the nodes may use a local, system-wide proxy server and only that proxy server needs to access the Sipwise license server.

### 11.3 How to Configure Licenses

The NGCP operator can set the **license key** in the main configuration file (`/etc/ngcp-config/config.yml`). The correct license key has to be entered in the configuration file, at the **general.license\_key** configuration parameter, so that licensing works as expected.

---

**Tip**

You always have to add the license key before being able to upgrade NGCP to release mr5.5.x or above. The upgrade script will look for the license key and will stop if it does not find the key.

---

The license key is also shown in the `/etc/ngcp-license-key` file once the key has been added to the configuration file and the new configuration has been applied.

---

**Note**

There is another configuration parameter related to licenses: `general.anonymous_usage_statistics` that has an effect on Sipwise NGCP CE installations only. This parameter enables / disables sending anonymous usage statistics to Sipwise.

---

Although not strictly related to NGCP configuration, the platform operator has to keep in mind that all NGCP nodes need to have **access to Sipwise license server**: `license.sipwise.com`

The operator has to ensure that there is no firewall rule or other network configuration that prevents NGCP nodes from connecting to Sipwise license server via HTTPS protocol (TCP, port 443).

### 11.4 How to Monitor License Client

As mentioned earlier in this chapter, the presence of license client can be monitored using the built-in utility "monit".

The other way to observe the behaviour of the license client is looking into the log file of "licensed" process: `/var/log/ngcp/licensed.log`

The NGCP operator may find entries like the below ones in case of normal operation:

```
Dec 12 16:20:42 sp1 ngcp-licensed[2205]: Valid license: [ABCDEFGH1_123456789_a1b2c3d4e5f6]:
  10000 calls, 1000000 subscribers, 2000000 registered subscribers, valid until Tue Jan 1
  00:00:00 2030 (signature valid until Tue Dec 26 16:20:43 2017)
Dec 12 16:22:41 sp1 ngcp-licensed[2205]: Usage report: 0 calls, 18 subscribers, 0 ←
  registered subscribers
```

where:

1. The first line shows *the licensed capacities*
2. The second line shows *the actual system usage indicators*

## 12 Software Upgrade

### 12.1 Release Notes

The sip:carrier version mr6.1.2 has the following important changes:

- Add timezone support to exported CDRs and time-based Call-Forwards [TT#27462, TT#24906]
- [CloudPBX] Add support to skip hunt-group members being on a call during call hunting [TT#6272]
- Improve Session Timer handling by adding a "transparent" mode and gracefully handle endpoints without SST support [TT#29931, TT#29926]
- Improve ngcp-panel memory usage and performance [TT#30507]
- Add mp3 and ogg formats support for fetching voicemail recordings via API

Please find the complete changelog in our release notes [on our WEB site](#).

### 12.2 Overview

The sip:carrier software upgrade procedure to mr6.1.2 will perform several fundamental tasks:

- Upgrade the NGCP software packages
- Upgrade the NGCP configuration templates
- Upgrade the NGCP DB schema
- Upgrade the NGCP configuration schema
- Upgrade the base system within Debian 9 (stretch) to the latest package versions

sip:carrier is a PRO-style system that has "A" and "B" sets of nodes with specific roles. The number of nodes can differ between installations and must be clarified before the upgrade at the planning stage.

The software upgrade is performed by Sipwise engineers with the following steps:

- software upgrade planning
- pre-upgrade steps: customtt, backups
- making all "B" nodes active
- ensure that all "A" nodes are standby
- upgrading all "A" nodes to the new release
- scheduling and performing a switchover to all "A" nodes

- ensuring that "A" nodes work well (otherwise, switch over back to "B" nodes)
- upgrading all "B" nodes to the new release
- performing system post-upgrade testing/cleanup

**Warning**

The only allowed software upgrade path is the one described above. All the other theoretically possible upgrade scenarios can lead to unpredictable results.

---

**Warning**

Nodes "A" and "B" MUST be used as described in this document. It is NOT allowed to swap them unless proxy replication (of MySQL on port 3308) is configured on the db01b node.

---

## 12.3 Planning a software upgrade

Have the following information prepared in writing:

- which system should be upgraded (clarify about LAB/LIVE, country, etc.)
- the date and time schedule for each of the steps above (keeping the time zone in mind)
- a confirmed timeframe for the upgrade operation (allowed switchover timeframe)
- a basic functionality test (BFT) to be executed before the start of the software upgrade and after the switchovers to ensure that the new release does not produce critical issues (the BFT should usually be prepared by the customer engineers)
- actions to be taken if the software upgrade operation cannot be completed within the defined maintenance window
- contact persons and ways of communication in case of emergency
- confirm the steps that will be performed in writing
- ensure that the customer and/or Sipwise engineers have access to the remote console of the servers: KVM, iDRAC, AMM

## 12.4 Preparing the software upgrade

It is recommended to execute all preparation steps in this chapter a few days before the actual software upgrade. They do not cause service downtime, so it is safe to execute them in peak hours.

### 12.4.1 Log into the standby management server (web01a/db01a).

---

**Tip**

Use the physical IP addresses instead of the shared one, so you can jump between the nodes later on.

---

Run the terminal multiplexer under the *sipwise* user (to reuse the Sipwise `.screenrc` settings that are convenient for working in multiple windows):

```
screen -S ngcp-upgrade
```

Become root inside your screen session:

```
sudo -s
```

Check the overall system status:

```
ngcp-status --all
```

Ensure that all the proxy nodes replicate the read-only DB (127.0.0.1:3308) from the db01a node. Otherwise, discuss a special plan to address this particularity.

For the following steps, investigate and make sure you understand why the custom modifications were introduced and if they are still required after the software upgrade. If the custom modifications are not required anymore, remove them (e.g. if a bug was fixed in the target release and the existing patch becomes irrelevant).

Create tickets to Sipwise developers to make relevant custom modifications part of the product in the future releases. This will allow you to get rid of the customtt files.

Find local changes to the template files by executing:

```
ngcp-customtt-diff-helper
```

`ngcp-customtt-diff-helper` can help you to download the templates for you target release. To do so, you can run the following command:

```
ngcp-customtt-diff-helper -d
```

In the `tmp` folder provided by the script you can, now, merge the current customtt with the new tt2, creating the new customtt.tt2 files. Once did that, you can TAR the new customtts in a tar.gz file and you can use that after the upgrade to deploy the new customtt.

```
ngcp-customtt-diff-helper -t
```

You can check further details regarding the `ngcp-customtt-diff-help` with the option `"-h"`.

Check if there are any `*.tt2.dpkg-dist` files among the templates. They usually appear when tt2 files are modified directly instead of creating customtt files. If you find any `*.tt2.dpkg-dist` files, then treat the corresponding tt2 files as if they were custom templates and introduce the changes from the existing tt2 files into the new templates (by creating associated `*.customtt.tt2`) before the software upgrade.

```
find /etc/ngcp-config -name \*.tt2.dpkg-dist
```

Note that in the end all `*.tt2.dpkg-dist` files must be removed before the software upgrade.



If you changed the configuration (e.g. added custom templates or changed the existing ones), then the system must be thoroughly tested when these changes are applied. Continue with the software upgrade preparation only when the results of the tests are successful.

Check and remove dpkg files left from previous software upgrades.

Make sure that the list is empty before you continue:

```
find /etc/ngcp-config -name \*.tt2.dpkg\*
```



#### Warning

If the installation uses locally specified mirrors, then the mirrors must be switched to the Sipwise APT repositories (at least for the software upgrade). Otherwise, the public Debian mirrors may not provide packages for old Releases anymore or at least provide outdated ones!

---

### 12.4.2 Log into the remaining servers

Open separate windows for all the servers inside your "screen" session. (Press `Ctrl+a + c` to open a new window, `Ctrl+a + a` or `Ctrl+a + [0-9]` to change the window. `Ctrl+a + "` shows the list of all your windows. Use `Ctrl+a + A` to change the window names to corresponding hosts).

Check the system for locally modified files (move them to appropriate `customtt.tt2` files if necessary) on **all** servers:

```
ngcp-status --integrity
```

Make sure the cluster status is OK: on **all** nodes manually run:

- **ngcpcfg status** - must print OK on all nodes

Check all nodes in parallel, using `clish` and `parallel-ssh`:

- **ngcp-clish "ngcp version summary"** - ensure that all cluster nodes have correct/expected from version
- **ngcp-clish "ngcp version package installed ngcp-ngcp-carrier"** - ensure that the metapackages version is equal to the ngcp version above
- **ngcp-clish "ngcp version package check"** - ensure that all nodes have the identical Debian package installed

---

#### Note

Software on all nodes must be identical before and after the upgrade!

---

- **ngcp-clish "ngcp cluster ssh connectivity"** - check SSH connectivity from the current node to all other nodes
  - **ngcp-clish "ngcp cluster ssh crossconnectivity"** - check SSH connectivity from all nodes to all other nodes
  - **ngcp-clish "ngcp monit summary"** - should not report any problems
-

- **ngcp-clish "ngcp cluster status"** - active nodes (with all services running) must print "all", the others must print "none"
- **ngcp-clish "ngcp status collective-check"** - should not report any problems
- **ngcp-clish "ngcp show date"** - date and time must be in sync on all the servers
- **ngcp-clish "ngcp show dns-servers"** - ensure that DNS records are correct

---

**Note**

to exit from *ngcp-clish* press Ctrl+Z (or type *exit*):

---

```
root@web01b:~# ngcp-clish
Entering 'clish-enable' view (press Ctrl+Z to exit)...
# exit
root@web01b:~#
```

Run "apt-get update" on all nodes. Ensure that you have no warnings/errors here.

Test the cluster failover to see if everything works fine on "B" nodes as well. On all the standby nodes execute:

```
ngcp-make-active
```

Afterwards, check *ngcp-status --all* again.

## 12.5 Upgrading the sip:carrier

Log in to all nodes and execute the checks from Section 12.4 again. This will ensure that nothing was broken since the preparation steps were finished. Also, execute **ngcpcfg show** and **ngcpcfg status** to check the latest configuration changes.

Perform the BFT test.

### 12.5.1 Preparing for maintenance mode

Sipwise NGCP introduces **Maintenance Mode** with its mr5.4.1 release. The maintenance mode of NGCP will disable some background services (for instance: *mediator*) during the software upgrade. It thus prevents the system from getting into an inconsistent state while the upgrade is being performed. You can activate maintenance mode by applying a simple configuration change as described later.

- Pull pending configuration (if any):

```
ngcpcfg pull
```

- Enable maintenance mode:

```
ngcpcfg set /etc/ngcp-config/config.yml "general.maintenance=yes"
```

- Apply configuration changes by executing:

```
ngcpcfg apply 'Enabling maintenance mode before the upgrade to mr6.1.2'  
ngcpcfg push all
```

To upgrade the sip:carrier to mr6.1.2 release, execute the following commands on the **standby management "A" node**:

### 12.5.2 Upgrading ONLY the first standby management node "A" (web01a/db01a)

---

#### Note

Sometimes the DB and MGMT roles are assigned to the same host. This is OK.

---



#### Warning

Do NOT execute the software upgrade on web01a and db01a in parallel!

---

The main goal of the following commands is to download the new packages into the approx cache. So all the nodes in the cluster will get identical packages.

```
NGCP_CURRENT_VERSION=$(cat /etc/ngcp_version)  
sed -i "s/$NGCP_CURRENT_VERSION/mr6.1.2/" /etc/apt/sources.list.d/sipwise.list  
  
ngcp-approx-cache-helper --auto --node localhost  
  
apt-get update  
apt-get install ngcp-upgrade-pro
```

---

#### Note

Don't worry, ngcp-upgrade-carrier does not exist, use ngcp-upgrade-pro as outlined above.

---



#### Warning

Do not use "ngcpcfg apply/build" after executing the steps from the above section, otherwise the changes will be overwritten and you will have to redo these steps. The same applies to similar sections below.

---

Execute ngcp-upgrade on the standby node as *root*:

```
ngcp-upgrade
```

---

#### Note

sip:carrier can be upgraded to mr6.1.2 from previous release or previous build only. The script ngcp-upgrade will find all the possible destination releases for the upgrade and allow to choose the proper one.

---

**Note**

If there is an error during the upgrade, the `ngcp-upgrade` script will request you to solve it. Once you've fixed the problem, just execute `ngcp-upgrade` again and it will continue from the previous step.

Merge/add the custom configuration templates if needed.

Apply the changes to configuration templates:

```
ngcpcfg apply 'applying customtt for new release mrX.X on node xxx01a'
```

Send the new templates to the shared storage and the other nodes

```
ngcpcfg push --nobuild --noapply all
```

**Warning**

Do NOT execute `ngcpcfg push --shared-only` at this stage, as it will affect further upgrades due to noticed outdated local `ngcpcfg` storage. If you did so, run `ngcpcfg push --nobuild --noapply all` once again to pull `ngcpcfg` changes on all the nodes from `glusterfs`.

**12.5.3 Upgrading the standby database node "A" (db\*a)****Note**

If the DB and MGMT roles are assigned to the same host, then skip this step as you have already upgraded the standby MGMT node "A" above.

Run the following commands to upgrade the standby DB node "A" (select the same release version as above and follow the on-screen recommendations):

```
NGCP_CURRENT_VERSION=$(cat /etc/ngcp_version)
sed -i "s/$NGCP_CURRENT_VERSION/mr6.1.2/" /etc/apt/sources.list.d/sipwise.list
apt-get update
apt-get install ngcp-upgrade-pro
ngcp-upgrade
```

**Note**

It is important to upgrade db01a node *before* upgrading any proxy nodes. Otherwise, the "local" MySQL (127.0.0.1:3308) on proxy nodes may become out of sync in case the new release has `_not_replicated.up` DB statements.

**12.5.4 Upgrading other standby nodes "A" (lb\*a/prx\*a)**

Run the below commands selecting the same release version and follow the on-screen recommendations:

```
NGCP_CURRENT_VERSION=$(cat /etc/ngcp_version)
sed -i "s/$NGCP_CURRENT_VERSION/mr6.1.2/" /etc/apt/sources.list.d/sipwise.list
apt-get update
apt-get install ngcp-upgrade-pro
ngcp-upgrade
```

### 12.5.5 Promote ALL standby nodes "A" to active.



#### Warning

Ensure that all standby nodes "A" are: \* upgraded to the new release (check `/etc/ngcp_version` or use `ngcp-clish`)  
\* have been rebooted (run `ngcp-status` on each standby node)

---

On all "A" nodes run:

```
ngcp-make-active
```

Ensure that the "A" nodes became active, by executing the `'ngcp-status'` and `'ngcp-clish'` commands described above.

Ensure that ALL "B" nodes are standby now!

### 12.5.6 Upgrading ALL standby nodes "B" (web\*b/db\*b/lb\*b/prx\*b)

Run the following commands selecting the same release version and following the on-screen recommendations:

```
NGCP_CURRENT_VERSION=$(cat /etc/ngcp_version)
sed -i "s/$NGCP_CURRENT_VERSION/mr6.1.2/" /etc/apt/sources.list.d/sipwise.list
apt-get update
apt-get install ngcp-upgrade-pro
ngcp-upgrade
```

---

#### Note

You can upgrade all standby "B" nodes simultaneously (including the ones with the mgmt and db roles).

---

## 12.6 Post-upgrade steps

### 12.6.1 Disabling maintenance mode

In order to disable the *maintenance mode*, do the following:

- Pull outstanding `ngcpcfg` changes (if any):

```
ngcpcfg pull
```

- Disable the maintenance mode:

```
ngcpcfg set /etc/ngcp-config/config.yml "general.maintenance=no"
```

- Apply the changes to configuration templates:

```
ngcpcfg apply 'Disable the maintenance mode after the upgrade to mr6.1.2'  
ngcpcfg push all
```

## 12.6.2 Post-upgrade checks

When everything has finished successfully, check that replication is running. Check `ngcp-status --all`. Finally, do a basic functionality test. Check the web interface, register two test subscribers and perform a test call between them to ensure call routing works.

---

### Note

You can find a backup of some important configuration files of your existing installation under `/var/backup/ngcp-mr6.1.2-*` (where `*` is a place holder for a timestamp) in case you need to roll back something at any time. A log file of the upgrade procedure is available at `/var/backup/ngcp-mr6.1.2-*/upgrade.log`.

---

## 12.7 Applying the Latest Hotfixes

If your current release is already the latest or you prefer to be on the LTS release, we still suggest applying the latest hotfixes and critical bug fixes.

Execute all steps as described in [?]. They include the system checks, customtt handling and others. It is important to execute all the steps from the above chapter.

It is suggested to promote B-nodes to active and start the update with A-nodes.

### 12.7.1 Update the approx cache on the standby management node

The main goal of the following command is to download the new packages into the approx cache. So all the nodes in the cluster will get identical packages.

```
ngcp-approx-cache-helper --auto --node localhost
```

### 12.7.2 Apply hotfixes on the standby management node

```
ngcp-update
```

### 12.7.3 Apply hotfixes on all other standby nodes

```
ngcp-update
```

### 12.7.4 Promote the standby nodes to active

Execute on the **standby** nodes as *root*:

```
ngcp-make-active
```

Check in a minute that the nodes became active:

```
ngcp-check-active
```

### 12.7.5 Apply hotfixes on new standby nodes

```
ngcp-update
```

Execute the final checks as described in the **Post-upgrade checks** section.

## 13 Backup, Recovery and Database Maintenance

### 13.1 sip:carrier Backup

For any service provider it is important to maintain a reliable backup policy as it enables prompt services restoration after any force majeure event. Although the design of sip:carrier implies data duplication and high availability of services, we still strongly suggest you to configure a backup procedure. The sip:carrier has a built-in solution that can help you back up the most crucial data. Alternatively, it can be integrated with any Debian compatible backup software.

#### 13.1.1 What data to back up

- The database

This is the most important data in the system. All subscriber and billing information, CDRs, user preferences, etc. are stored in the MySQL server. It is strongly recommended to have up-to-date dumps of all the databases on corresponding NGCP nodes.

- System configuration

The system configuration files such as */etc/mysql/sipwise.cnf* and the */etc/ngcp-config/* directory should be included in the backup as well. We suggest backing up the whole */etc* folder.

- Exported CDRs (optional)

The */home/jail/home/cdreexport* directory contains the exported CDRs. It depends on your call data retention policy whether or not to remove these files after exporting them to an external system.

#### 13.1.2 The built-in backup solution

The sip:carrier comes with an easy-to-use solution that creates everyday backups of the most important data:

- The system configuration files. The whole */etc* directory is backed up.
- Exported CDRs. The */home/jail/home/cdreexport* directory with csv files.
- All required databases on corresponding servers.

This functionality is disabled by default and can be enabled and configured in the *backuptools* subsection in the *config.yml* file. Please, refer to the “C.1.3 backup tools” section of the “NGCP configs overview” chapter for the backup configuration options.

Once you set the required configuration options, apply the changes:

```
ngcpcfg apply 'enabled the backup feature'  
ngcpcfg push all
```



Once you activate the feature, the sip:carrier will create backups in the off-peak time on the standby nodes and put them to the `/var/backup/ngcp_backup` directory. You can copy these files to your backup server using `scp` or `ftp`.

---

**Note**

make sure that you have enough free disk space to store the backups for the specified number of days.

---

## 13.2 Recovery

In the worst case scenario, when the system needs to be recovered from a total loss, you only need 4 steps to get the services back online:

- Install the sip:carrier as explained in chapter 2.
- Restore the `/etc/ngcp-config/` directory and the `/etc/mysql/sipwise.cnf` file from the backup, overwriting your local files.
- Restore the database from the latest MySQL dump.
- Apply the changes to bring the original configuration into effect:

```
ngcpcfg apply 'restored the system from the backup'  
ngcpcfg push all
```

## 13.3 Reset Database

**Important**

All existing data will be wiped out! Use this script only if you want to clear all previously configured services and start configuration from scratch.

---

To reset database to its original state you can use a script provided by CE: \* Execute `ngcp-reset-db`. It will assign new unique passwords for the NGCP services and reset all services. The script will also create dumps for all NGCP databases.

## 13.4 Accounting Data (CDR) Cleanup

Sipwise sip:carrier offers an easy way to cleanup, backup or archive old accounting data—i.e. CDRs—that is not necessary for further processing any more, or must be deleted according to the law. There are some NGCP components designed for this purpose and they are commonly called *cleantools*. These are basically configurable scripts that interact with NGCP's `accounting` and `kamailio` databases, or remove exported CDR files in order to clean or archive the unnecessary data.

### 13.4.1 Cleanuptools Configuration

The configuration parameters of *cleanuptools* are located in the main NGCP configuration file: `/etc/ngcp-config/config.yml`. Please refer to the `config.yml` file description: [Cleanuptools Configuration Data](#) Section [B.1.8](#) for configuration parameter details.

In case the system administrator needs to modify some configuration value, the new configuration must be activated in the usual way, by running the following commands:

```
> ngcpcfg apply 'Modified cleanuptools config'
> ngcpcfg push all
```

As a result new configuration files will be generated for the accounting database and the exported CDR cleanup tools. Please read detailed description of those tools in subsequent sections of the handbook.

The NGCP system administrator can also select the time when cleanup scripts are run, by modifying the schedule here: `/etc/cron.d/cleanup-tools`

### 13.4.2 Accounting Database Cleanup

The script responsible for cleaning up the database is: `/usr/sbin/acc-cleanup.pl`

The configuration file used by the script is: `/etc/ngcp-cleanup-tools/acc-cleanup.conf`

An extract from a sample configuration file is provided here:

```
#####

batch = 10000
archive-target = /var/backup/cdr
compress = gzip

username = dbcleaner
password = rcKamRdHhx7saYRbkJfP
host = localhost

connect accounting
time-column = from_unixtime(start_time)
backup-months = 2
backup-retro = 2
backup cdr

connect accounting
archive-months = 2
archive cdr

connect kamailio
```

```
time-column = time
cleanup-days = 90
cleanup acc

# Clean up after mediator by deleting old leftover acc entries and deleting
# old entries out of acc_trash and acc_backup
connect kamailio
time-column = time
cleanup-days = 30
cleanup acc_trash
cleanup acc_backup
```

The configuration file itself contains a detailed description of how database cleanup script works. It consists of a series of statements, one per line, which are going to be executed in sequence. A statement can either just set a variable to some value, or perform an action.

There are 3 types of actions the database cleanup script can take:

- backup CDRs
- archive CDRs
- cleanup CDRs

These actions are discussed in following sections.

A generic action is connecting to the proper database: `connect <database name>`

### 13.4.2.1 Backup CDRs

The database cleanup tool can create *monthly backups* of CDRs in the `accounting` database and store those data records in separate tables named: `cdr_YYYYMM`. The instruction in the configuration file looks like: `backup <table name>`, by default and typically it is: `backup cdr`

Configuration values that govern the backup procedure are:

- `time-column`: Which column in `cdr` table shows the month which a CDR belongs to.
- `batch`: How many records to process within a single SQL statement. If unset, less than or equals 0, all of them are processed at once.
- `backup-months`: How many months worth of records to keep in the `cdr` table—where current CDRs are stored—and not move into the monthly backup tables.



#### Important

Months are always processed as a whole, thus the value specifies how many months to keep AT MOST. In other words, if the script is started on December 15th and this value is set to "2", then all of December and November is kept, and all of October will be backed up.

---

- `backup-retro`: How many months to process for backups, going backwards in time. Using the example above, with this value set to "3", the months October, September and August would be backed up, while any older records would be left untouched.

### 13.4.2.2 Archive CDRs

The database cleanup tool can archive (dump) old monthly backup tables. The statement used for this purpose is: `archive <table name>`, by default and typically it is: `archive cdr`

This creates an SQL dump out of too old tables created by the `backup` statement and drop them afterwards from database. Archiving uses the following configuration values:

- `archive-months`: Uses the same logic as the `backup-months` variable above. If set to "12" and the script was started on December 15th, it will start archiving with the December table of the previous year.



#### Important

Note that the sum of `backup-retro` + `backup-months` values cannot be larger than `archive-months` value for the same table. Otherwise you end up creating empty monthly backup tables, only to dump and delete them right afterwards.

---

- `archive-target`: Target directory for writing the SQL dump files into. If explicitly specified as `"/dev/null"`, then no actual archiving will be performed, but instead the tables will only be dropped from database.
- `compress`: If set to "gzip", then gzip the dump files after creation. If unset, do not compress.
- `host`, `username` and `password`: As dumping is performed by an external command, those variables are reused from the `connect` statement.

### 13.4.2.3 Cleanup CDRs

The database cleanup tool may do database table cleanup without performing backup. In order to do that, the statement: `clean up <table name>` is used. Typically this has to be done in `kamailio` database, examples:

- `cleanup acc`
- `cleanup acc_trash`
- `cleanup acc_backup`

Basically the `cleanup` statement works just like the `backup` statement, but doesn't actually backup anything, but rather just deletes old records. Configuration values used by the procedure:

- `time-column`: Gives the database column name that shows the time of CDR creation.
- `batch`: The same as with `backup` statement.
- `cleanup-days`: Any record older than this many days will be deleted.

### 13.4.3 Exported CDR Cleanup

The script responsible for cleaning up exported CDR files is: `/usr/sbin/cleanup-old-cdr-files.pl`

The configuration file used by exported CDR cleanup script is: `/etc/ngcp-cleanup-tools/cdr-files-cleanup.yml`

A sample configuration file is provided here:

```
enabled: no
max_age_days: 30
paths:
-
  path: /home/jail/home/*/20[0-9][0-9][0-9][0-9]/[0-9][0-9]
  wildcard: yes
  remove_empty_directories: yes
  max_age_days: ~
-
  path: /home/jail/home/cdrexpert/resellers/*/20[0-9][0-9][0-9][0-9]/[0-9][0-9]
  wildcard: yes
  remove_empty_directories: yes
  max_age_days: ~
-
  path: /home/jail/home/cdrexpert/system/20[0-9][0-9][0-9][0-9]/[0-9][0-9]
```

The exported CDR cleanup tool simply deletes CDR files in the directories provided in the configuration file, if those have already expired.

Configuration values that define the files to be deleted:

- `enabled`: Enable (yes) or disable (no) exported CDR cleanup.
- `max_age_days`: Gives the expiration time of the exported CDR files in days. There is a general value which may be overridden by a local value provided at a specific path. The local value is valid for the particular path only.
- `paths`: an array of path definitions
  - `path`: a path where CDR files are to be found and deleted; this may contain wildcard characters
  - `wildcard`: Enable (yes) or disable (no) using wildcards in the path
  - `remove_empty_directories`: Enable (yes) or disable (no) removing empty directories if those are found in the given path
  - `max_age_days`: the local expiration time value for files in the particular path

## 14 Platform Security, Performance and Troubleshooting

Once the sip:carrier is in production, security and maintenance becomes really important. In this chapter, we'll go through a set of best practices for any production system.

### 14.1 Sipwise SSH access to sip:carrier

The sip:carrier provides SSH access to the system for Sipwise operational team for debugging and final tuning. Operational team uses user *sipwise* which can be logged in through SSH key only (password access is disabled) from dedicated access server *jump.sipwise.com* only.

To completely remove Sipwise access to your system, please execute as user root:

```
root@myserver:~# ngcp-support-access --disable && apt-get install ngcp-support-noaccess
```

---

#### Note

you have to execute the command above on each node of your sip:carrier system!

---



#### Warning

please ensure that the script complete successfully:

---

```
* Support access successfully disabled.
```

If you need to restore Sipwise access to the system, please execute as user root:

```
root@myserver:~# apt-get install ngcp-support-access && ngcp-support-access --enable
```



#### Warning

please ensure that the script complete successfully:

---

```
* Support access successfully enabled.
```

### 14.2 Firewalling

#### 14.2.1 Firewall framework

The sip:carrier runs a wide range of services. In order to secure the platform while allowing access to the sip:carrier, the NGCP configuration framework provides a set of predefined network zones. Services are aggregated into appropriate zones by default. Zones are assigned to network interfaces (and VLANs if applicable) in `/etc/ngcp-config/network.yml`.

**Caution**

Though the default firewall setup provided by the NGCP configuration framework provides a safe setup for sip:carrier, security audits of the platform performed by qualified engineers before commissioning the platform into service are strongly recommended. Customization of the setup requires in-depth knowledge of firewalling principles in general and the *netfilter* facility in particular.

Table 19: NGCP network zones

Zone name	Description
ha_int	Internal cluster interface providing internal cluster communications between cluster pairs (heartbeat) and synchronization of data and configuration
mon_ext	Interface to connect external monitoring appliances (SNMP)
rtp_ext	Interface for external RTP media relay between sip:carrier and endpoints (e.g. user agents, peers)
sip_ext	Interface for external SIP signalling between sip:carrier and endpoints (e.g. user agents, peers)
sip_int	Interface for internal signalling, e.g. between load-balancers, proxies and applications servers
ssh_ext	Interface providing external access to the sip:carrier command line interface
web_ext	Interface providing access to the customers' self-care Web panel
web_int	Interface for access to the administrative Web panel, its REST APIs and internal API communications

**Note**

Additional custom zones may be configured, but will not be automatically integrated into the firewall configuration.

To facilitate firewall functionality, sip:carrier uses the Kernel's *netfilter* facility and *iptables-persistent* as an interface to *netfilter*. *Netfilter* is using *tables* and within that *chains* to store rules in this hierarchy: *table* → *chain* → *rule*. Default firewall setups of sip:carrier do not use netfilter tables *nat* and *raw*, but only default table *filter*.

**Note**

Custom *nat* rules for IPv4 and IPv6 may be added in file `/etc/ngcp-config/config.yml` in sections `security→firewall→nat_rules4` and `security→firewall→nat_rules6`.

Each *chain* deploys a *default policy* handling packets which did not trigger and rule in a particular *chain*.

Table 20: NGCP *netfilter* default policies

Chain	Default policy	Description
INPUT	DROP	Handling all packets directly destined for a sip:carrier node (only packets matching a rule are allowed)
FORWARD	DROP	Handling all packets received by a sip:carrier node and destined for another, non-local IP destination (no default rules added)
OUTPUT	ACCEPT	Handling all packets originating on a sip:carrier node (no default rules added)
rtpengine	N/A	Container for rtpengine rule to allow the rule to persist even when the Kernel module is unloaded (e.g. during upgrades)

The default firewall setup provided by sip:carrier:

- adds rules to INPUT to secure access to platform and services
- blocks all traffic from and to FORWARD
- allows all OUTPUT traffic

#### 14.2.2 NGCP firewall configuration

The sip:carrier comes with a preconfigured set of firewall rules, which can be enabled and configured in `/etc/ngcp-config/config.yml` in section `security→firewall`. Refer to Section [B.1.32](#) for available configuration options.

Firewall configuration is applied by running `ngcpcfg apply`. However, this will not activate new rules automatically to avoid inadvertent self-lockout. To finally activate new firewall rules run `iptables-apply`. This will prompt for another system logon to verify access remains available. If the prompt is not confirmed, firewall rules will automatically be reverted to the previous state re-enabling access to the command line.

#### Caution



The NGCP firewall subsystem by default is disabled in `/etc/ngcp-config/config.yml` key `security.firewall.enable: no`. This is to avoid blocking any traffic inadvertently during installation. After the firewall subsystem has been configured appropriately, it needs to be enabled by setting `security.firewall.enable: yes` in `/etc/ngcp-config/config.yml`.

#### 14.2.3 IPv4 System rules

The following set of rules is added by the system upon activation of the firewall subsystem. Individual system rules are configured in `/etc/ngcp-config/templates/etc/iptables/rules.v4.tt2` and `/etc/ngcp-config/templates/etc/iptables/rules.v6.tt2`



Table 21: Firewall system rules

Zone	Chain	Target	Rule	Description
all	INPUT	rtpengine	<code>-p udp -j rtpengine</code>	Redirects all incoming UDP packets to chain <i>rtpengine</i> (putting RTPENGINE rule into a dedicated chain allows for the rule to persist even when the Kernel module gets unloaded, e.g. during upgrades)
all	rtpengine	RTPENGINE	<code>-p udp -j RTPENGINE --id 0</code>	Feeds all RTP packets to RTPENGINE Kernel module
n/a	INPUT	ACCEPT	<code>-i lo -j ACCEPT</code>	Accept all packets received by local loopback interface
all	INPUT	ACCEPT	<code>-m state --state RELATED,ESTABLISHED -j ACCEPT</code>	Accept all incoming packets tied to <i>related</i> or <i>established</i> connections
all	INPUT (IPv4)	ACCEPT	<code>-p icmp -m icmp --icmp-type 8 -j ACCEPT</code>	Accept all ICMP <i>echo</i> messages
all	INPUT (IPv4)	ACCEPT	<code>-p icmp -m icmp --icmp-type 0 -j ACCEPT</code>	Accept all ICMP <i>echo reply</i> messages
all	INPUT (IPv6)	ACCEPT	<code>-A INPUT -p ipv6-icmp -j ACCEPT</code>	Accept all ICMPv6 messages
all	INPUT	cluster	<code>-j cluster</code>	Divert all incoming packets to the <i>cluster</i> chain
all	cluster	ACCEPT	<code>-s &lt;node_ip&gt; -j ACCEPT</code>	Set of rules white-listing all IP-addresses owned by the NGCP platform for incoming traffic
api_int	INPUT	ACCEPT	<code>-p tcp --dport &lt;ossbss.port&gt; -j ACCEPT</code>	Set of rules for all <i>api_int</i> interfaces accepting all incoming packets for API port defined in <i>/etc/ngcp-config/config.yml</i> with key <i>ossbss.port</i>
mon_ext	INPUT	ACCEPT	<code>+p udp -s &lt;snmpclient_ip&gt; --dport 161 -j ACCEPT</code>	Set of rules for all <i>mon_ext</i> interfaces based on a list of IPs for all SNMP communities configured in <i>checktools.snmpd.communities</i>
rtp_ext	INPUT	ACCEPT/ <i>name</i>	<code>-p udp --dport &lt;rtpproxy.minport&gt;: '&lt;rtpproxy.maxport&gt;' -j ACCEPT/<i>name</i></code>	Set of rules for all <i>rtp_ext</i> interfaces accepting all incoming packets for RTP port range defined in <i>/etc/ngcp-config/config.yml</i> with keys <i>rtpproxy.minport</i> and <i>rtpproxy.maxport</i> (see note below for custom options)

Table 21: (continued)

Zone	Chain	Target	Rule	Description
sip_ext	INPUT	ACCEPT	<code>-p udp --dport &lt;kamailio.lb.port&gt; -j ACCEPT</code>	Set of rules for all <i>sip_ext</i> interfaces accepting all packets on the loda balancer's SIP signalling port defined in <i>/etc/ngcp-config/config.yml</i> with key <i>kamailio.lb.port</i> (UDP)
sip_ext	INPUT	ACCEPT	<code>-p tcp --dport &lt;kamailio.lb.port&gt; -j ACCEPT</code>	Set of rules for all <i>sip_ext</i> interfaces accepting all packets on the loda balancer's SIP signalling port defined in <i>/etc/ngcp-config/config.yml</i> with key <i>kamailio.lb.port</i> (TCP)
sip_ext	INPUT	ACCEPT	<code>-p tcp --dport &lt;kamailio.lb.tls.port&gt; -j ACCEPT</code>	Set of rules for all <i>sip_ext</i> interfaces accepting all packets on the loda balancer's SIP signalling port defined in <i>/etc/ngcp-config/config.yml</i> with key <i>kamailio.lb.tls.port</i> (TCP/TLS)
sip_ext	INPUT	ACCEPT	<code>-p tcp --dport 5222 -j ACCEPT</code>	Set of rules for all <i>sip_ext</i> interfaces accepting all packets on TCP port 5222 (XMPP client)
sip_ext	INPUT	ACCEPT	<code>-p tcp --dport 5269 -j ACCEPT</code>	Set of rules for all <i>sip_ext</i> interfaces accepting all packets on TCP port 5269 (XMPP server)
sip_ext	INPUT	ACCEPT	<code>-p tcp --dport &lt;pushd. port&gt; -j ACCEPT</code>	Set of rules for all <i>sip_ext</i> interfaces accepting all packets incoming for the <i>pushd</i> server port configured in <i>/etc/ngcp-config/config.yml</i> with key <i>pushd.port</i>
ssh_ext	INPUT	ACCEPT	<code>-A INPUT -i &lt;ssh_ext_interface&gt; -p tcp -s &lt;sshd. permit_support_from&gt; - -dport sshd.port -j ACCEPT</code>	List of rules to accept incoming packets for SSH on all <i>ssh_ext</i> interfaces from hosts configured in <i>/etc/ngcp-config/config.yml</i> with key <i>sshd.permit_support_from</i>

Table 21: (continued)

Zone	Chain	Target	Rule	Description
web_ext	INPUT	ACCEPT	<pre>-p tcp --dport &lt;www_admin.http_csc. port&gt; -j ACCEPT</pre>	List of rules to accept incoming packets for the <i>Customer Self Care</i> interface defined in <i>/etc/ngcp-config/config.yml</i> with key <i>www_admin.http_csc.port</i> on all <i>web_ext</i> interfaces
web_int	INPUT	ACCEPT	<pre>-p tcp --dport &lt;www_admin.http_admin. port&gt; -j ACCEPT</pre>	List of rules to accept incoming packets for the <i>Admin Panel</i> interface defined in <i>/etc/ngcp-config/config.yml</i> with key <i>www_admin.http_admin.port</i> on all <i>web_int</i> interfaces

**Caution**

To function correctly, the *rtengine* requires an additional *iptables* rule installed. This rule (with a target of `RTPENGINE`) is automatically installed and removed when the *rtengine* starts and stops, so normally you don't need to worry about it. However, any 3rd party firewall solution can potentially flush out all existing *iptables* rules before installing its own, which would leave the system without the required `RTPENGINE` rule and this would lead to decreased performance. It is imperative that any 3rd party firewall solution either leaves this rule untouched, or installs it back into place after flushing all rules out. The complete parameters to install this rule (which needs to go into the `INPUT` chain of the `filter` table) are: `-p udp -j RTPENGINE --id 0`

**Note**

Some of the parameters used to populate the firewall rules automatically may contain hostnames instead of IP addresses. Since firewall rules need to be configured based on IP addresses by design, the NGCP configuration framework will lookup such hostnames during *ngcpcfg apply* and expand them to the IP addresses as returned by *gethostbyname*. If DNS resolving changes for such hostnames due to changes to DNS the rules will not update automatically. Another run of *ngcpcfg apply* will be needed to reperform the lookup and update the rules to reflect changes in DNS. If this step is omitted, clients may be locked out of the system.

**Note**

By default, the rules for the `rtplib_ext` zone are created with a target of `ACCEPT`. It is optionally possible to create these rules with another `iptables` chain as target, and instruct the RTP proxy to dynamically manage individual rules for each running call in this chain. If this is enabled, the chain with the name given in the `/etc/ngcp-config/config.yml` key `rtplib_ext→firewall_iptables_chain` will be created as empty, leaving the effective target for UDP packets within the RTP port range as the table's default policy (normally `DROP`). The RTP proxy will then dynamically create one `ACCEPT` rule for each open RTP media port in the given chain when a call starts, and delete it when the call is finished. It should be noted that dynamically creating and deleting `iptables` rules can incur a significant performance overhead, especially in scenarios with high call volumes, and it is therefore not recommended to enable this feature in such cases.

**14.2.4 Custom rules**

The NGCP configuration framework allows to add custom rules to the firewall setup in `/etc/ngcp-config/config.yml`. The custom rules are added after the system rules. Hence, they apply for packets not matched by the system rules only.

Example custom rule to whitelist all IPv4 traffic from network interface `eth1.301` effectively making VLAN 301 a trusted network:

```
rules4:
  - '-A INPUT -i eth1.301 -j ACCEPT'
```

Example custom rule to accept incoming traffic from monitoring station `203.0.113.93` for an optionally installed `check_mk` agent:

```
rules4:
  - '-A INPUT -p tcp -s 203.0.113.93 --dport 6556 -j ACCEPT'
```

To add hosts or networks to the SSH whitelist they can be either added to key `sshd.permit_support_from` in `/etc/ngcp-config/config.yml` or a custom rule may be used:

```
rules4:
  - '-A INPUT -s 198.51.100.0/24 --dport 22 -j ACCEPT'
  - '-A INPUT -s 203.0.113.93 --dport 22 -j ACCEPT'
```

**Note**

In custom rules keys from `/etc/ngcp-config/config.yml` cannot be referenced. Thus, the values need to be manually looked up, hard coded, and kept in sync manually. This is by design of YAML.

**14.2.5 Example firewall configuration section**

An example for NGCP firewall configuration in `/etc/ngcp-config/config.yml` enabling both the firewall subsystem and the logging facility may look like:

```
security:
  firewall:
    enable: 'yes'
```

```
logging:
  enable: 'yes'
  file: '/var/log/firewall.log'
  tag: 'NGCPFW'
policies:
  input: 'DROP'
  forward: 'DROP'
  output: 'ACCEPT'
rules4:
  - '-A INPUT -i eth0 -j ACCEPT'
```

## 14.3 Password management

The sip:carrier comes with some default passwords the user should change during the deployment of the system. They have been explained in the previous chapters of this handbook.



### Important

Many NGCP services use MySQL backend. Users and passwords for these services are created during the installation. These passwords are unique for each installation, and the connections are restricted to localhost. You should not change these users and passwords.

---

### 14.3.1 The "root" account

The sip:carrier's super-user account comes with a preconfigured password. It is imperative that this password is changed by the operator immediately after the sip:carrier is shipped and before it is connected to any potentially unsecure public or private network using a secure password in compliance with existing password policies of the operator. The "root" password must not be shared outside of the operator's organization including Sipwise engineers. The "root" password must not be shared in any publicly accessible communications including e-mail or ticketing systems.

To change the root password log into the freshly deployed system as "root" using the preconfigured password and execute:

```
root@myserver:~# passwd
```

Then follow the prompts to change the password.

### 14.3.2 The "administrator" account

The sip:carrier Web-interface comes with a preconfigured "administrator" account deployed with a default password. This account can be considered the NGCP application super-user and has far-reaching access to application specific settings via the Web-interface. It is imperative that the password for this account is changed by the operator immediately after the sip:carrier is shipped and before it is connected to any potentially unsecure public or private network using a secure password in compliance with existing password policies of the operator. The "administrator" password must not be shared outside of the operator's organization including

Sipwise engineers. The "administrator" password must not be shared in any publicly accessible communications including e-mail or ticketing systems.

The password for the "administrator" account can be changed via the Web-interface.

### 14.3.3 The "cdreexport" account

The login for the system account *cdreexport* is disabled by default. Although this is a jailed account, it has access to sensitive information, namely the Call Detail Records of all calls. SSH keys should be used to login this user, or alternatively a really strong password should be used when setting the password via *passwd cdreexport*.

### 14.3.4 The MySQL "root" user

The *root* user in MySQL has no default password. A password should be set using the *mysqladmin password* command.

### 14.3.5 The "ngcpsoap" account

Generate new password for user *ngcpsoap* to access the provisioning interfaces, see the details in Section 8.

## 14.4 SSL certificates.

The sip:carrier provides default, self-signed SSL certificates for SSL connections. These certificates are common for every installation. Before going to production state, the system administrator should provide SSL certificates for the web services. These certificates can either be shared by all web interfaces (*provisioning*, *administrator interface* and *customer self care interface*), or separate ones for each them can be used.

- Generate the certificates. The *customer self care interface* certificate should be signed by a certification authority to avoid browser warnings.
- Upload the certificates to the system
- Set the path to the new certificates in */etc/ngcp-config/config.yml*:
  - *ossbss→apache→autoprov→sslcertfile* and *ossbss→apache→autoprov→sslcertkeyfile* for the *provisioning interface*.
  - *ossbss→apache→restapi→sslcertfile* and *ossbss→apache→restapi→sslcertkeyfile* for the *REST interface*.
  - *www\_admin→http\_admin→sslcertfile* and *www\_admin→http\_admin→sslcertkeyfile* for the *admin interface*.
  - *www\_admin→http\_csc→sslcertfile* and *www\_admin→http\_csc→sslcertkeyfile* for the *customer self care interface*.
- Apply the configuration changes with *ngcpcfg apply 'added web ssl certs'*.

The sip:carrier also provides the self-signed SSL certificates for SIP over TLS services. The system administrator should replace them with certificates signed by a trusted certificate authority if he is going to enable it for the production usage (*kamailio→lb→tls→enable* (disabled by default)).

- Generate the certificates.
- Upload the certificates to the system
- Set the path to the new certificates in `/etc/ngcp-config/config.yml`:
  - `kamailio→lb→tls→sslcrtfile` and `kamailio→lb→tls→sslcrtkeyfile` .
- Apply the configuration changes with `ngcpcfg apply 'added kamailio certs'`.

## 14.5 Securing your sip:carrier against SIP attacks

The sip:carrier allows you to protect your VoIP system against SIP attacks, in particular **Denial of Service** and **brute-force attacks**. Let's go through each of those attacks and let's see how to configure your system in order to face such situations and react against them.

### 14.5.1 Denial of Service

As soon as you have packets arriving on your sip:carrier server, it will require a bit of time of your CPU. Denial of Service attacks are aimed to break down your system by sending floods of SIP messages in a very short period of time and keep your system busy to handle such huge amount of requests. sip:carrier allows you to block such kind of attacks quite easily, by configuring the following section in your `/etc/ngcp-config/config.yml` :

```
security:
  dos_ban_enable: 'yes'
  dos_ban_time: 3600
  dos_reqs_density_per_unit: 50
  dos_sampling_time_unit: 2
  dos_whitelisted_ips: []
  dos_whitelisted_subnets: []
```

Basically, as soon as sip:carrier receives more than 50 messages from the same IP in a time window of 2 seconds, that IP will be blocked for 3600 sec, and you will see in the the `kamailio-lb.log` a line saying:

```
Nov 9 00:11:53 sp1 lb[41958]: WARNING: <script>: IP '1.2.3.4' is blocked and banned - R=< ↔
null> ID=304153-3624477113-19168@tedadg.testlab.local
```

The banned IP will be stored in kamailio memory, you can check the list via web interface or via the following command:

```
# ngcp-kamctl lb fifo sht_dump ipban
```



#### Important

You have to run this command on ACTIVE load balancer node.

---

## Excluding SIP endpoints from banning

There may be some SIP endpoints that send a huge traffic towards NGCP from a specific IP address. A typical example is a *SIP Peering Server*.



### Caution

sip:carrier supports handling such situations by excluding all defined *SIP Peering Servers* from DoS protection mechanism.

---

The NGCP platform administrator may also add whitelisted IP addresses manually in `/etc/ngcp-config/config.yml` at `kamailio.lb.security.dos_whitelisted_ips` and `kamailio.lb.security.dos_whitelisted_subnets` parameters.

## 14.5.2 Bruteforcing SIP credentials

This is a very common attack you can easily detect checking your `/var/log/ngcp/kamailio-proxy.log`. You will see INVITE/REGISTER messages coming in with strange usernames. Attackers is trying to spoof/guess subscriber's credentials, which allow them to call out. The very first protection against these attacks is: **ALWAYS USE STRONG PASSWORD**. Nevertheless sip:carrier allow you to detect and block such attacks quite easily, by configuring the following `/etc/ngcp-config/config.yml` section:

```
failed_auth_attempts: 3
failed_auth_ban_enable: 'yes'
failed_auth_ban_time: 3600
```

You may increase the number of failed attempt if you want (in some cases it's better to be safed, some users can be banned accidentally because they are not writing the right password) and adjust the ban time. If a user try to authenticate an INVITE (or REGISTER) for example and it fails more then 3 times, the "user@domain" (not the IP as for Denial of Service attack) will be block for 3600 seconds. In this case you will see in your `/var/log/ngcp/kamailio-lb.log` the following lines:

```
Nov 9 13:31:56 sp1 lb[41952]: WARNING: <script>: Consecutive Authentication Failure for ' ←
sipvicous@mydomain.com' UA='sipvicous-client' IP='1.2.3.4' - R=<null> ID ←
=313793-3624525116-589163@testlab.local
```

Both the banned IPs and banned users are shown in the Admin web interface, you can check them by accessing the **Security Bans** section in the main menu. You can check the banned user as well by retrieving the same info directly from kamailio memory, using the following commands:

```
# ngcp-kamctl lb fifo sht_dump auth
```



### Important

You have to run this command on ACTIVE load balancer node.

---



## 14.6 Topology Hiding

### 14.6.1 Introduction to Topology Hiding on NGCP

The term "topology hiding" in SIP is used to describe the measures taken by typically an SBC (Session Border Controller) to hide detailed information of the internal network at the border of which it is located. Pieces of information such as IP addresses and port numbers used by SIP endpoints and intermediaries within the network are considered sensitive, as these can give some hints to potential attackers about the topology of the network.

In a typical SIP session the mandatory headers may carry that sensitive information, for example: *Contact*, *Via*, *Record-Route*, *To*, *From*, *Call-ID*. An SBC applying topology hiding will mangle the content of those headers.

Concealment of sensitive information is achieved through encoding the original content of selected SIP headers. Then NGCP will create a new SIP URI using a preselected IP address and the encoded content as URI parameter, finally re-assembling the SIP header.

Examples for encoded SIP headers:

```
Record-Route: <sip:127.0.0.8;line=sr-NvaAlWtecghucEhu6WtAcu...>
Contact: <sip:127.0.0.8;line=sr-NvaAli-1VeL.kRxLcbN86W...>
```

The *load-balancer* element of the Sipwise NGCP has an SBC role, from the SIP peers point of view. The *LB* offers topology hiding function that can be simply activated through a configuration change. By default the function is disabled.

### 14.6.2 Configuration of Topology Hiding

Activating topology hiding function is possible through the modification of the following configuration parameters in `/etc/ngcp-config/config.yml` file (shown below with default values of parameters):

```
kamailio:
  lb:
    security:
      topoh:
        enable: no
        mask_callid: no
        mask_ip: 127.0.0.8
```

Meaning of the configuration parameters:

- `enable`: if set to `yes`, the topology hiding will be activated
- `mask_callid`: if set to `yes`, the SIP Call-ID header will also be encoded
- `mask_ip`: an IP address that will be used to create valid SIP URIs, after encoding the real/original header content.

---

**Tip**

Any valid, preferably private network address can be used. The suggestion is however to use an address that is not used by any other SIP endpoint or intermediary element in the network.

---

### 14.6.3 Considerations for Topology Hiding

Although hiding sensitive information about a SIP provider's network is desired, there are some potential side effects caused by topology hiding.

The most common example is the consequence that **SIP message size may grow** when applying topology hiding. The fact that SIP messages become larger may even prevent NGCP from communicating successfully with another SIP entity (a peer SBC, for example). This can be expected under following circumstances:

- SIP transport protocol is UDP
- SIP messages have more *Via* and *Record-Route* headers
- IP packets of SIP messages without the topology hiding feature already have a size close to the MTU

In such a case the IP packets carrying SIP messages with encoded headers will have a size exceeding the MTU, that will cause loss of data.

The recommended solution in such a case is to use TCP transport for SIP messages.

## 14.7 System Requirements and Performance

The sip:carrier is a very flexible system, capable of serving from hundreds to several tens of thousands of subscribers in a single node. The system comes with a default configuration, capable of serving up to 50.000 subscribers in a *normal* environment. But there is no such thing as a *normal* environment. And the sip:carrier has sometimes to be tuned for special environments, special hardware requirements or just growing traffic.

---

### Note

If you have performance issues with regards to disk I/O please consider enabling the *noatime* mount option for the root filesystem. Sipwise recommends the usage of *noatime*, though remove it if you use software which conflicts with its presence.

---

In this section some parameters will be explained to allow the sip:carrier administrator tune the system requirements for optimum performance.

Table 22: Requirement\_options

Option	Default value	Requirement impact
cleantools→binlog_days	15	Heavy impact on the harddisk storage needed for mysql logs. It can help to restore the database from backups or restore broken replication.
database→bufferpoolsize	64MB	For test systems or low RAM systems, lowering this setting is one of the most effective ways of releasing RAM. The administrator can check the innodb buffer hit rate on production systems; a hit rate over 99% is desired to avoid bottlenecks.

Table 22: (continued)

Option	Default value	Requirement impact
<code>kamailio→lb→pkg_mem</code>	16	This setting affects the amount of RAM the system will use. Each kamailio-lb worker will have this amount of RAM reserved. Lowering this setting up to 8 will help to release some memory depending on the number of kamailio-lb workers running. This can be a dangerous setting as the lb process could run out of memory. Use with caution.
<code>kamailio→lb→shm_mem</code>	1/16 * Total System RAM	The installer will set this value to 1/16 of the total system RAM. This setting does not change even if the system RAM does so it's up to the administrator to tune it. It has been calculated that 1024 (1GB) is a good value for 50K subscriber environment. For a test environment, setting the value to 64 should be enough. "Out of memory" messages in the kamailio log can indicate that this value needs to be raised.
<code>kamailio→lb→tcp_children</code>	8	Number of TCP workers kamailio-lb will spawn per listening socket. The value should be fine for a mixed UDP-TCP 50K subscriber system. Lowering this setting can free some RAM as the number of kamailio processes would decrease. For a test system or a pure UDP subscriber system 2 is a good value. 1 or 2 TCP workers are always needed.
<code>kamailio→lb→tls→enable</code>	yes	Enable or not TLS signaling on the system. Setting this value to "no" will prevent kamailio to spawn TLS listening workers and free some RAM.
<code>kamailio→lb→udp_children</code>	8	See <code>kamailio→lb→tcp_children</code> explanation
<code>kamailio→proxy→children</code>	8	See <code>kamailio→lb→tcp_children</code> explanation. In this case the proxy only listens udp so these children should be enough to handle all the traffic. It could be set to 2 for test systems to lower the requirements.
<code>kamailio→proxy→*_expires</code>		Set the default and the max and min registration interval. The lower it is more REGISTER requests will be handled by the lb and the proxy. It can impact in the network traffic, RAM and CPU usage.
<code>kamailio→proxy→natping_interval</code>	30	Interval for the proxy to send a NAT keepalive OPTIONS message to the nated subscriber. If decreased, this setting will increase the number of OPTIONS requests the proxy needs to send and can impact in the network traffic and the number of natping processes the system needs to run. See <code>kamailio→proxy→natping_processes</code> explanation.
<code>kamailio→proxy→natping_processes</code>	7	Kamailio-proxy will spawn this number of processes to send keepalive OPTIONS to the nated subscribers. Each worker can handle about 250 messages/second (depends on the hardware). Depending the number of nated subscribers and the <code>kamailio→proxy→natping_interval</code> parameter the number of workers may need to be adjusted. The number can be calculated like $\text{nated\_subscribers}/\text{natping\_interval}/\text{pings\_per\_second\_per\_process}$ . For the default options, assuming 50K nated subscribers in the system the parameter value would be $50.000/30/250 = (6,66) 7$ workers. 7 is the maximum number of processes kamailio will accept. Raising this value will cause kamailio not to start.

Table 22: (continued)

Option	Default value	Requirement impact
kamailio→proxy→shm_mem	1/16 * Total System RAM	See <i>kamailio→lb→shm_mem</i> explanation.
rateomat→enable	yes	Set this to no if the system shouldn't perform rating on the CDRs. This will save CPU usage.
rsyslog→external_log	0	If enabled, the system will send the log messages to an external server. Depending on the <i>rsyslog→external_loglevel</i> parameter this can increase dramatically the network traffic.
rsyslog→ngcp_logs_preserve_days	93	This setting will set the number of days ngcp logs under <i>/var/log/ngcp</i> will be kept in disk. Lowering this setting will free a high amount of disk space.

**Tip**

In case of using virtualized environment with limited amount of hardware resources, you can use the script *ngcp-toggle-performance-config* to adjust sip:carrier configuration for high/low performance:

```
root@spce:~# /usr/sbin/ngcp-toggle-performance-config
/usr/sbin/ngcp-toggle-performance-config - tool to adjust sip:provider configuration for ↔
low/high performance

--help           Display this usage information
--high-performance Adjust configuration for system with normal/high performance
--low-performance Adjust configuration for system with low performance (e.g. VMs)

root@spce:~#
```

**14.8 Troubleshooting**

The sip:carrier platform provides detailed logging and log files for each component included in the system via rsyslog. The main folder for log files is */var/log/ngcp/*, it contains a list of self explanatory log files named by component name.

The sip:carrier is a high performance system which requires compromise between traceability (maximum amount of debug information being written to hard drive) and productivity (minimum load on IO subsystem). This is the reason why different log levels are configured for the provided components by default.

Most log files are designed for debugging sip:carrier by Sipwise operational team while main log files for daily routine usage are:

<b>Log file</b>	<b>Content</b>	<b>Estimated size</b>
/var/log/ngcp/api.log	API logs providing type and content of API requests and responses as well as potential errors	medium
/var/log/ngcp/panel.log /var/log/ngcp/panel-debug.log	Admin Web UI logs when performing operational tasks on the ngcp-panel	medium
/var/log/ngcp/cdr.log	mediation and rating logs, e.g. how many CDRs have been generated and potential errors in case of CDR generation or rating fails for particular accounting data	medium

Log file	Content	Estimated size
/var/log/ngcp/ha.log	fail-over related logs in case a node in a pair loses connection to the other side, when a standby node takes over or an active node goes standby due to intra-node communication issues or external ping node connection issues	small
/var/log/ngcp/kamailio-proxy.log	Overview of SIP requests and replies between lb, proxy and sems processes. It's the main log file for SIP overview	huge
/var/log/ngcp/kamailio-lb.log	Overview of SIP requests and replies along with network source and destination information flowing through the platform	huge

Log file	Content	Estimated size
/var/log/ngcp/sems.log	Overview of SIP requests and replies between lb, proxy and sems processes	small
/var/log/ngcp/rtp.log	rtpengine related log, showing information about RTP communication	small

**Warning**

it is highly NOT recommended to change default log levels as it can cause system IO overloading which will affect call processing.

**Note**

the exact size of log files depend on system type, system load, system health status and system configuration, so cannot be estimated with high precision. Additionally operational network parameters like ASR and ALOC may impact the log files' size significantly.

**14.8.1 Collecting call information from logs**

The easiest way to fetch information about a single call among the log files is the search for the SIP CallID (a unique identifier for a SIP dialog). The call ID is used as call marker in almost all the voip related log file, such as `/var/log/ngcp/kamailio-lb.log` , `/var/log/ngcp/kamailio-proxy.log` , `/var/log/ngcp/sems.log` or `/var/log/ngcp/rtp.log`. Example of kamailio-proxy.log line:

```
Nov 19 00:35:56 sp1 proxy[7475]: NOTICE: <script>: New request on proxy - M=REGISTER R=sip: ←
sipwise.local
F=sip:jdoe@sipwise.local T=sip:jdoe@sipwise.local IP=10.10.1.10:5060 (127.0.0.1:5060) ID ↔
=364e4676776621034977934e055d19ea@127.0.0.1 UA='SIP-UA 1.2.3.4'
```

The above line shows the SIP information you can find in a general line contained in `/var/log/ngcp/kamailio-*`:

- M=REGISTER : The SIP Method
- R=sip:sipwise.local : The SIP Request URI
- F=sip:jdoe@sipwise.local : The SIP From header

- T=sip:jdoe@sipwise.local : The SIP To header
- IP=10.10.1.10:5060 (127.0.0.1:5060) : The source IP where the message is coming from. Between brackets it is shown the local internal IP where the message come from (in this case Load Balancer)
- ID=364e4676776621034977934e055d19ea@127.0.0.1 : The SIP CallID.
- UAIP=10.10.1.10 : The User Agent source IP
- UA=SIP-UA 1.2.3.4 : The SIP User Agent header

In order to collect the full log related to a single call, it's necessary to "grep" the `/var/log/ngcp/kamailio-proxy.log` using the `ID=` string, for example:

```
# grep "364e4676776621034977934e055d19ea@127.0.0.1" /var/log/ngcp/kamailio-proxy.log
```

### 14.8.2 Collecting SIP traces

The sip:carrier platform provides several tools to collect SIP traces. It can be used the sip:carrier `ngrep-sip` tool to collect SIP traces, for example to fetch traffic in text format from outbound and among load balancer, proxy and sems :

```
# ngrep-sip b
```

see the manual to know all the options:

```
# man ngrep-sip
```

The `ngrep` debian tool can be used in order to make a SIP trace and save it into a `.pcap` file :

```
# ngrep -s0 -Wbyline -d any -O /tmp/SIP_trace_file_name.pcap port 5062 or port 5060
```

The `sngrep` debian graphic tool as well can be used to visualize SIP trace and save them in a `.pcap` file :

```
# sngrep
```



## 15 Monitoring and Alerting

### 15.1 Internal Monitoring

#### 15.1.1 Process monitoring via monit

The platform uses the internal *monit* service to monitor all essential services. Since the sip:carrier runs in an active/standby mode, not all services are always running on both nodes, some of them will only run on the active node and be stopped on the standby node. The following commands show the most critical services on the platform: `* monit summary` - to get the list of services and their current status, `* monit status` - to get the list of services with detailed status.



#### Important

When you perform a stop/start/monitor/unmonitor operation on a service, *monit* affects other services that depend on the initial one. Hence, if you stop or unmonitor a service all services that depend on it will be stopped or unmonitored as well.

---

For example, `monit stop mysql` operation will stop kamailio, sbc, asterisk, prosody and some other services. Although the recommended way to operate on services is via the `ngcp-services` wrapper which will take care of abstracting the underlying process monitoring implementation.

If any service ever fails for whatever reason the *monit* daemon quickly restarts it. When that happens, the daemon will send a notification email to the address specified in the `config.yml` file under the `general.adminmail` key. It will also send warning emails to this address under certain abnormal conditions, such as high memory consumption (> 75% is used) or high CPU load.



#### Important

In order for *monit* to be able to send emails to the specified address, the local MTA (*exim4*) must be configured correctly. If you haven't done so already, run `dpkg-reconfigure exim4-config` to do this. The CE edition's handbook contains more information about this in the *Installation* chapter.

---

#### 15.1.2 System monitoring via Telegraf

The platform uses the internal *telegraf* service to monitor many aspects of the system, including CPU, memory, swap, disk, filesystem, network, processes, NTP, Nginx, Redis and MySQL.

The gathered information is stored in *InfluxDB*, in the *telegraf* database.

#### 15.1.3 NGCP-specific monitoring via ngcp-witnessd

The platform uses the internal *ngcp-witnessd* service to monitor NGCP-specific metrics or system metrics currently not tracked by *telegraf*, including memory, process count, Heartbeat, MTA, Kamailio, SIP and MySQL.

The gathered information is stored in *InfluxDB*, in the *ngcp* database.

#### 15.1.4 Monitoring data in InfluxDB

The platform uses *InfluxDB* as a time series database, to store most of the metrics collected in the system.

On a sip:carrier each node stores its own metrics and the ones for their peer node, and the management nodes store the metrics for all the nodes in the cluster. This is done via *influxdb-relay* which listens for *InfluxDB* writes and multiplexes them to the local node and any other node necessary.

The monitoring data is used by various components of the platform, including *ngcp-collective-check*, *ngcp-snmp-agent* and by the statistics dashboard powered by *Grafana*.

The monitoring data can also be accessed directly by various means; by using the *influx* command-line tool in CLI or TUI modes; by using the *ngcp-influxdb-extract* wrapper which provides two convenience commands to run arbitrary queries or to fetch the last value for a measurement's field; or by using the HTTP API with *curl* (or other HTTP fetchers), or with the *Sipwise::InfluxDB::HTTP* perl module.

See [https://docs.influxdata.com/influxdb/v1.1/query\\_language/spec/](https://docs.influxdata.com/influxdb/v1.1/query_language/spec/) for information about InfluxQL, the query language used by *InfluxDB*.

---

**Tip**

To get the list of all measurements for a specific database the following query can be used `SHOW MEASUREMENTS`.

---

---

**Tip**

To get the list of fields for a specific measurement the following query can be used `SELECT LAST(*) FROM "measurement"`.

---

---

**Tip**

To get the list of tags for a specific measurement the following query can be used `SHOW TAG KEYS FROM "measurement"`, and for all the current tag values for a tag `SHOW TAG VALUES FROM "measurement" WITH KEY = "tag"`.

---

See Section [F.2.1](#) for detailed information about the list of data currently stored in the *InfluxDB ngcp* monitoring database.

## 15.2 Statistics Dashboard

The platform's administration interface (described in Section [4](#)) provides a graphical overview based on *Grafana* of the most important system health indicators, such as memory usage, load averages and disk usage. VoIP statistics, such as the number of concurrent active calls, the number of provisioned and registered subscribers, etc. is also present.

## 15.3 External Monitoring Using SNMP

### 15.3.1 Overview and Initial Setup

The sip:carrier exports a variety of cluster health data and statistics over the standard SNMP interface. By default, the SNMP interface can only be accessed locally. To make it possible to provide the SNMP data to an external system, the `config.yml` file needs to be edited and the list of allowed community names and allowed hosts/IP ranges must be populated. This list can be found under the `checktools.snmpd.communities` key and it consists of one or more `community/source` value pairs. The `community` is the allowed community name, while `source` is an IP address or an IP block where to allow the requests from.

The SNMP notifications can also be configured in a similar way, to send them to an external system, by populating the `checktools.snmpd.trap_communities` key with `community/target` value pairs. The `community` is the value that will be used when sending the trap, while the `target` is an IP address where to send the trap.

The `public` entries with the `localhost` source and target are used for local testing of SNMP functionality. It is recommended that you leave these entries in place. Other legal `sources` can be formed as single IP addresses or IP blocks in IP/prefix notation, for example `192.168.115.0/24`. Other `targets` can be formed as single IP addresses.

---

#### Tip

To locally check if SNMP is working correctly, execute the command `snmpwalk -v2c -cpublic localhost .` (note the trailing dot). This will generate a long list of raw SNMP OIDs and their values, provided that the `default` SNMP community key has been left in place.

---

---

#### Tip

To locally check if SNMP notifications (or traps) are working correctly, install the `snmptrapd` package, which will be configured by default to catch the traps sent by the `localhost` SNMP agent. The traps will show up on `/var/log/daemon.log`, and a couple of traps can be generated simply by running `service snmpd restart`.

---

INFO: SNMP version 1 and version 2c are supported.

### 15.3.2 Details

There are two types of information that can be retrieved from SNMP. The first one is the native NGCP cluster overview from the Sipwise MIBs (Management Information Bases). The second is the legacy ad-hoc information using the Net-SNMP extension OIDs, and detailed information for the node running the SNMP daemon using standard OIDs (Object Identifiers).

#### 15.3.2.1 Sipwise NGCP OIDs

The entire NGCP cluster can be monitored by using the `SIPWISE-NGCP-MIB`, `SIPWISE-NGCP-MONITOR-MIB` and `SIPWISE-NGCP-STATS-MIB`. These OIDs are rooted at the Sipwise NGCP slot `.1.3.6.1.4.1.34274.1.*`.

The MIBs are self-documented, and can be found as part of the *ngcp-snmp-mibs* package (running `dpkg -S SIPWISE*MIB` will list their pathnames). The NGCP SNMP Agent is a part of the *ngcp-snmp-agent* package, which is installed by default and works out-of-the-box as long as the `snmpd` has been properly configured.

The `SIPWISE-NGCP-MIB` acts as the root MIB and provides information about the cluster licensing and layout (which is mostly static data about each node, such as node name, its IP address, its roles, etc.) and information required to access the OIDs from the other MIBs.

The `SIPWISE-NGCP-MONITOR-MIB` provides current monitoring information, global health conditions, the number of provisioned and registered subscribers and devices. It also provides per node information (independently of the number of nodes or their names) on their filesystem, processes, databases, system load, memory, heartbeat status, MTA queues, etc.

The `SIPWISE-NGCP-STATS-MIB` provides accumulated statistics on billing, performance and processed SIP messages.

NOTICE: OIDs under the following trees are not yet implemented: *ngcpMonitorFraud*, *ngcpMonitorPerformance.perfCAPSCurTable* and *ngcpStats*.

INFO: The NGCP SNMP Agent uses *Redis* and *InfluxDB* as data sources. This data is essential for accurate and complete monitoring data in the SNMP OID tree. In addition, the *Redis* database must be available on a shared IP address, so that *ngcp-witnessd* can always write to it.

### 15.3.2.2 Legacy OIDs

---

#### Note

The following OIDs have been superseded by the Sipwise NGCP OIDs, but they are still provided for backwards compatibility.

---

All basic system health variables (such as memory, disk, swap, CPU usage, network statistics, process lists, etc.) for the *mgmt* node can be found in standard OID slots from standard MIBs. For example, memory statistics can be found through the *UCD-SNMP-MIB* in OIDs such as `memTotalSwap.0`, `memAvailSwap.0`, `memTotalReal.0`, `memAvailReal.0`, etc., which translate to numeric OIDs `.1.3.6.1.4.1.2021.4.*`. In fact, *UCD-SNMP-MIB* is the most useful MIB for overall system health checks.

Additionally, there's a list of specially monitored processes, also found through the *UCD-SNMP-MIB*. `UCD-SNMP-MIB::prNames (.1.3.6.1.4.1.2021.2.1.2)` gives the list of monitored processes, `prCount (.1.3.6.1.4.1.2021.2.1.5)` is how many of each process are running and `prErrorFlag (.1.3.6.1.4.1.2021.2.1.100)` gives a 0/1 error indication (with `prErrMsg (.1.3.6.1.4.1.2021.2.1.101)` providing an explanation of any error).

---

#### Tip

Some of these processes are not supposed to be running on the standby node, so you'll see the error flag raised there. A possible solution is to run these SNMP checks against the shared service IP of the cluster.

---

Furthermore, *UCD-SNMP-MIB* provides a list of custom external checks. The names of these can be found under the `UCD-SNMP-MIB::extNames (.2)` tree, with `extOutput (.101)` providing the output (one line) from each check and `extResult (.100)` the exit code from each check.

The first of these external checks called `collective_check` provides a combined and overall system health status indicator. It gathers information from both nodes and returns `0` in `extResult.1 (.100.1)` if everything is OK and running as it should. If it finds a problem somewhere, but with the system still operational (e.g. a service is stopped on the inactive node), `extResult.1` will return `1` and `extOutput.1` will be set to a string that can be used to diagnose the problem. In case the system is found in a critical and non-operational state, `extResult.1` will return `2`, again with an error message set. If you want to keep it really simple, you can just monitor this one OID and raise an alarm if it ever goes to non-zero.

INFO: The `0/1/2` status codes allow for easy integration with *Nagios*.

The remaining external checks simply return statistics on the system, they all return a number in `extOutput` and have `extResult` always set to zero.

The full list of such checks is below. All of these checks have three modes: the first returns the statistics from `sp1` (the first node in the sip:carrier pair), the second - from `sp2`, and the third - from whichever node is being queried (which is useful when querying the shared service IP). For example, the local SIP response time from `sp1` is in `sip_check_sp1`, from `sp2` - is in `sip_check_sp2`, and from the host itself - is in `sip_check_self`.

The base OID of the Result and Output OIDs is always `.1.3.6.1.4.1.2021.8.1`, so if you read `.100.1`, the full OID is `.1.3.6.1.4.1.2021.8.1.100.1`.

Name in MIB	Result OID	Output OID	Name	Description
UCD-SNMP-MIB::extNames.1	.100.1	.101.1	collective_check	Summarized platform check
UCD-SNMP-MIB::extNames.2	.100.2	.101.2	sip_check_sp1	SIP response time in seconds on sp1
UCD-SNMP-MIB::extNames.3	.100.3	.101.3	sip_check_sp2	SIP response time in seconds on sp2
UCD-SNMP-MIB::extNames.4	.100.4	.101.4	mysql_check_sp1	Average number of MySQL queries per second on sp1
UCD-SNMP-MIB::extNames.5	.100.5	.101.5	mysql_check_sp2	Average number of MySQL queries per second on sp2
UCD-SNMP-MIB::extNames.6	.100.6	.101.6	mysql_replication_check_sp1	MySQL replication delay in seconds on sp1
UCD-SNMP-MIB::extNames.7	.100.7	.101.7	mysql_replication_check_sp2	MySQL replication delay in seconds on sp2
UCD-SNMP-MIB::extNames.8	.100.8	.101.8	mpt_check_sp1	RAID status on sp1
UCD-SNMP-MIB::extNames.9	.100.9	.101.9	mpt_check_sp2	RAID status on sp2
UCD-SNMP-MIB::extNames.10	.100.10	.101.10	exim_queue_check_sp1	Number of mails undelivered in MTA queue on sp1

Name in MIB	Result OID	Output OID	Name	Description
UCD-SNMP-MIB::extNames.11	.100.11	.101.11	exim_queue_check_sp2	Number of mails undelivered in MTA queue on sp2
UCD-SNMP-MIB::extNames.12	.100.12	.101.12	provisioned_subscribers_sp1	Number of subscribers provisioned on sp1
UCD-SNMP-MIB::extNames.13	.100.13	.101.13	provisioned_subscribers_sp2	Number of subscribers provisioned on sp2
UCD-SNMP-MIB::extNames.14	.100.14	.101.14	kam_dialog_active_checks_sp1	Number of active calls on sp1
UCD-SNMP-MIB::extNames.15	.100.15	.101.15	kam_dialog_active_checks_sp2	Number of active calls on sp2
UCD-SNMP-MIB::extNames.16	.100.16	.101.16	kam_dialog_early_checks_sp1	Number of calls in Early Media state on sp1
UCD-SNMP-MIB::extNames.17	.100.17	.101.17	kam_dialog_early_checks_sp2	Number of calls in Early Media state on sp2
UCD-SNMP-MIB::extNames.18	.100.18	.101.18	kam_dialog_type_local_calls_sp1	Number of active calls local on sp1
UCD-SNMP-MIB::extNames.19	.100.19	.101.19	kam_dialog_type_local_calls_sp2	Number of active calls local on sp2
UCD-SNMP-MIB::extNames.20	.100.20	.101.20	kam_dialog_type_relay_calls_sp1	Number of active calls routed via peers on sp1
UCD-SNMP-MIB::extNames.21	.100.21	.101.21	kam_dialog_type_relay_calls_sp2	Number of active calls routed via peers on sp2
UCD-SNMP-MIB::extNames.22	.100.22	.101.22	kam_dialog_type_incoming_calls_sp1	Number of incoming calls on sp1
UCD-SNMP-MIB::extNames.23	.100.23	.101.23	kam_dialog_type_incoming_calls_sp2	Number of incoming calls on sp2
UCD-SNMP-MIB::extNames.24	.100.24	.101.24	kam_dialog_type_outgoing_calls_sp1	Number of outgoing calls on sp1
UCD-SNMP-MIB::extNames.25	.100.25	.101.25	kam_dialog_type_outgoing_calls_sp2	Number of outgoing calls on sp2
UCD-SNMP-MIB::extNames.26	.100.26	.101.26	kam_usrloc_regusers_checks_sp1	Number of subscribers with at least one active registration on sp1

Name in MIB	Result OID	Output OID	Name	Description
UCD-SNMP-MIB::extNames.27	.100.27	.101.27	kam_usrloc_regusers_check_sp2	Number of subscribers with at least one active registration on sp2
UCD-SNMP-MIB::extNames.28	.100.28	.101.28	kam_usrloc_regdevices_total_sp1	Total number of registered end devices on sp1
UCD-SNMP-MIB::extNames.29	.100.29	.101.29	kam_usrloc_regdevices_total_sp2	Total number of registered end devices on sp2
UCD-SNMP-MIB::extNames.30	.100.30	.101.30	mysql_replication_discrepancy_mysql1	Number of MySQL tables not in sync between sp1 and sp2
UCD-SNMP-MIB::extNames.31	.100.31	.101.31	mysql_replication_discrepancy_mysql2	Number of MySQL tables not in sync between sp1 and sp2
UCD-SNMP-MIB::extNames.32	.100.32	.101.32	sip_check_self	Summarized platform check on active node
UCD-SNMP-MIB::extNames.33	.100.33	.101.33	mysql_check_self	Average number of MySQL queries per second on active node
UCD-SNMP-MIB::extNames.34	.100.34	.101.34	mysql_replication_check_mysql	MySQL replication delay in seconds on active node
UCD-SNMP-MIB::extNames.35	.100.35	.101.35	mpt_check_self	RAID status on active node
UCD-SNMP-MIB::extNames.36	.100.36	.101.36	exim_queue_check_self	Number of mails undelivered in MTA queue on active node
UCD-SNMP-MIB::extNames.37	.100.37	.101.37	provisioned_subscribers_number_self	Number of subscribers provisioned on active node
UCD-SNMP-MIB::extNames.44	.100.44	.101.44	kam_usrloc_regusers_check_active	Number of subscribers with at least one active registration on active node
UCD-SNMP-MIB::extNames.45	.100.45	.101.45	kam_usrloc_regdevices_total_active	Total number of registered end devices on active node

Name in MIB	Result OID	Output OID	Name	Description
UCD-SNMP-MIB::extNames.46	.100.46	.101.46	mysql_replication_discrepancy	Number of MySQL tables not in sync between sp1 and sp2
UCD-SNMP-MIB::extNames.47	.100.47	.101.47	kam_dialog_type_local_checks	Number of active local calls on active proxy X
UCD-SNMP-MIB::extNames.48	.100.48	.101.48	kam_dialog_type_relay_checks	Number of active calls routed via peers on active proxy X
UCD-SNMP-MIB::extNames.49	.100.49	.101.49	kam_dialog_type_incoming_checks	Number of incoming calls on active proxy X
UCD-SNMP-MIB::extNames.50	.100.50	.101.50	kam_dialog_type_outgoing_checks	Number of outgoing calls on active proxy X
UCD-SNMP-MIB::extNames.51	.100.51	.101.51	kam_dialog_active_checks	Number of active calls on active proxy X
UCD-SNMP-MIB::extNames.52	.100.52	.101.52	kam_dialog_early_checks	Number of calls in Early Media state on active proxy X

---

**Tip**

Some of the checks can be disabled (most are enabled by default) through the `config.yml` file, and those checks will then return an error message or an empty string in their `extOutput`. Enable those checks in the config file to get their output in the SNMP OID tree. The enable/disable flags can be found in the `checktools` section.

---



## 16 Extensions and Additional Modules

### 16.1 Cloud PBX

The sip:carrier comes with a commercial Cloud PBX module to provide B2B features for small and medium sized enterprises. The following chapters describe the configuration of the PBX features.

#### 16.1.1 Configuring the Device Management

The *Device Management* is used by admins and resellers to define the list of device models, firmwares and configurations available for end customer usage. These settings are pre-configured for the default reseller up-front by Sipwise and have to be set up for every reseller separately, so a reseller can choose the devices he'd like to serve and potentially tweak the configuration for them. [List of available pre-configured devices](#) Section 16.1.13.

End customers choose from a list of *Device Profiles*, which are defined by a specific *Device Model*, a list of *Device Firmwares* and a *Device Configuration*. The following sections describe the setup of these components.

To do so, go to *Settings*→*Device Management*.

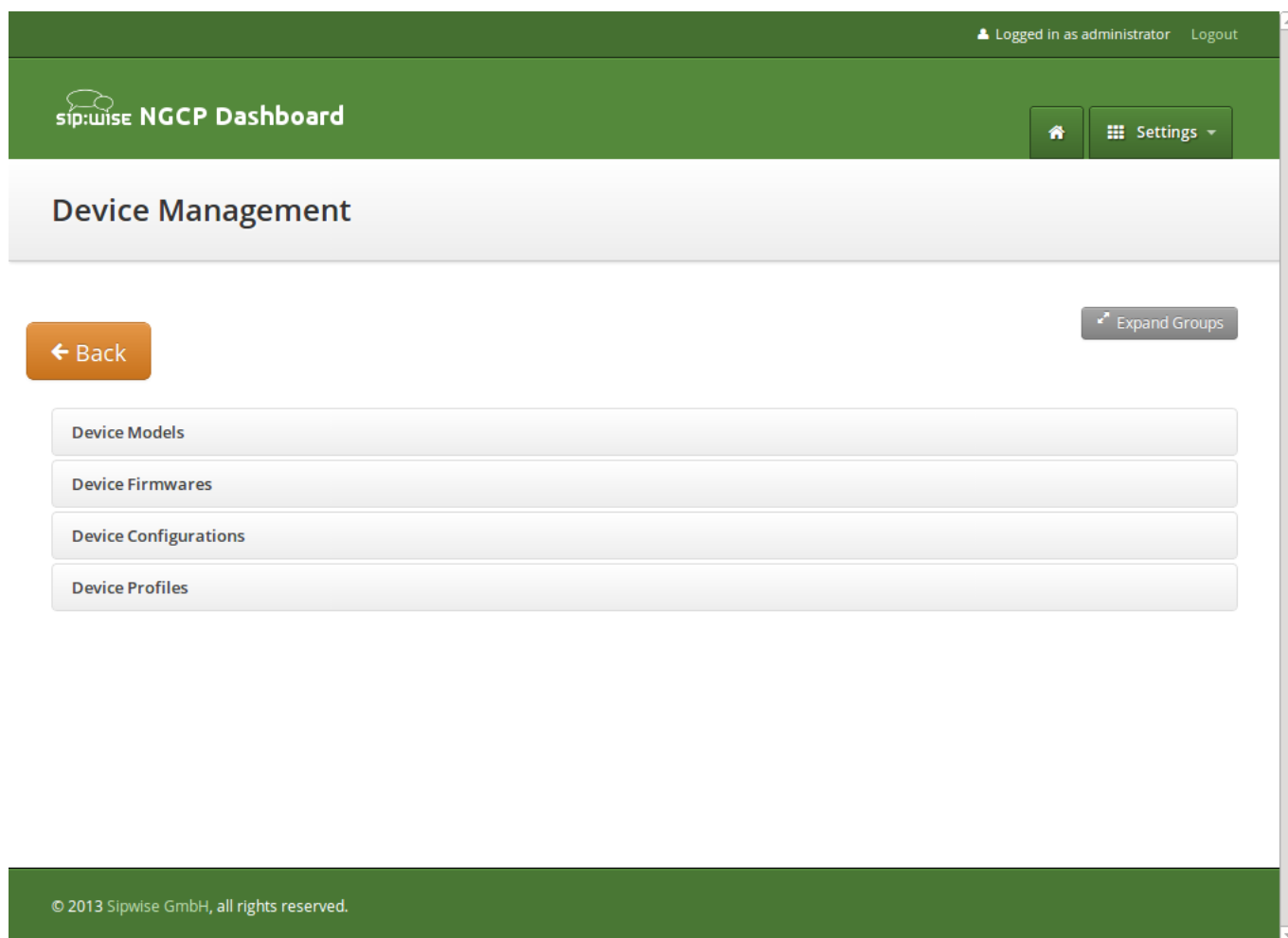


Figure 98: Device Management

#### 16.1.1.1 Setting up Device Models

A *Device Model* defines a specific hardware device, like the vendor, model name, the number of keys and their capabilities. For example a Cisco SPA504G has 4 keys, which can be used for private lines, shared lines (SLA) and busy lamp field (BLF). If you have an additional attendant console, you get 32 more buttons, which can only do BLF.

In this example, we will create a Cisco SPA504G with an additional Attendant Console.

Expand the *Device Models* row and click *Create Device Model*.

First, you have to select the reseller this device model belongs to, and define the vendor and model name.

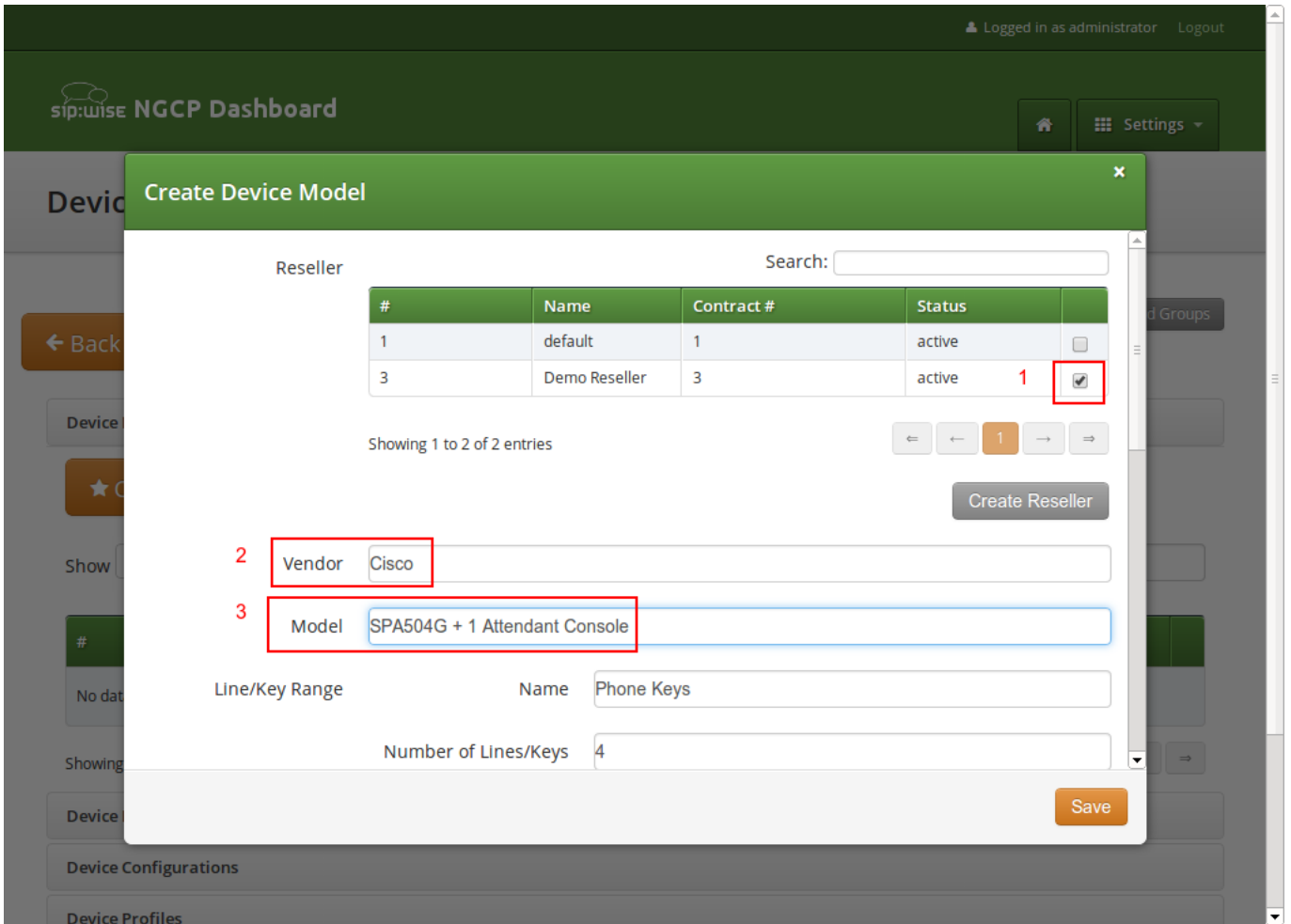


Figure 99: Create Device Model Part 1

In the *Line/Key Range* section, you can define the first set of keys, which we will label `Phone Keys`. The name is important, because it is referenced in the configuration file template, which is described in the following sections. The SPA504G internal phone keys support private lines (where the customer can assign a normal subscriber, which is used to place and receive standard phone calls), shared lines (where the customer can assign a subscriber which is shared across multiple people) and busy lamp field (where the customer can assign other subscribers to be monitored when they get a call, and which also acts as speed dial button to the subscriber assigned for BLF), so we enable all 3 of them.

The screenshot shows the 'Create Device Model' modal in the NGCP Dashboard. The form is for a Cisco SPA504G + 1 Attendant Console. The 'Line/Key Range' section is highlighted with a red box, showing a 'Name' field with the value 'Phone Keys' and a 'Number of Lines/Keys' field with the value '4'. Below this, three options are checked: 'Supports Private Line', 'Supports Shared Line', and 'Supports Busy Lamp Field'. The 'Remove' and 'Save' buttons are visible at the bottom right of the modal.

Figure 100: Create Device Model Part 2

In order to also configure the attendant console, press the *Add another Line/Key Range* button to specify the attendant console keys.

Again provide a name for this range, which will be `Attendant Console 1` to match our configuration defined later. There are 32 buttons on the attendant console, so set the number accordingly. Those 32 buttons only support BLF, so make sure to **uncheck** the private and shared line options, and only check the `busy lamp field` option.

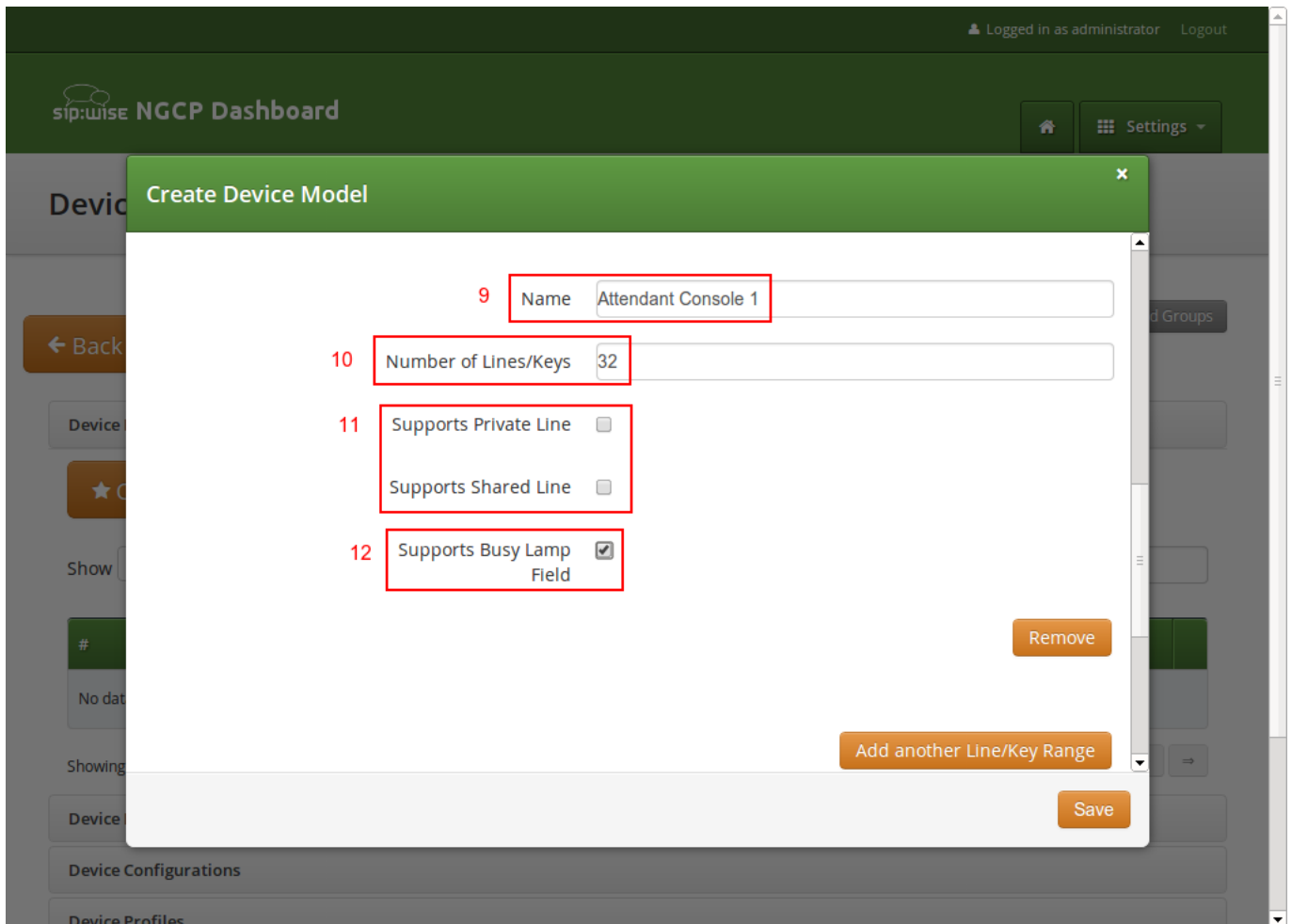


Figure 101: Create Device Model Part 3

The last two settings to configure are the *Front Image* and *MAC Address Image* fields. Upload a picture of the phone here in the first field, which is shown to the customer for him to recognize easily how the phone looks like. The MAC image is used to tell the customer where he can read the MAC address from. This could be a picture of the back of the phone with the label where the MAC is printed, or an instruction image how to get the MAC from the phone menu.

The rest of the fields are left at their default values, which are set to work with Cisco SPAs. Their meaning is as follows:

- *Bootstrap Sync URI*: If a stock phone is plugged in for the first time, it needs to be provisioned somehow to let it know where to fetch its configuration file from. Since the stock phone doesn't know about your server, you have to define an HTTP URI here, where the customer is connected with his web browser to set the according field.
- *Bootstrap Sync HTTP Method*: This setting defines whether an HTTP GET or POST is sent to the Sync URI.
- *Bootstrap Sync Params*: This setting defines the parameters appended to the Sync URI in case of a GET, or posted in the request body in case of POST, when the customer presses the *Sync* button later on.

Finally press *Save* to create the new device model.

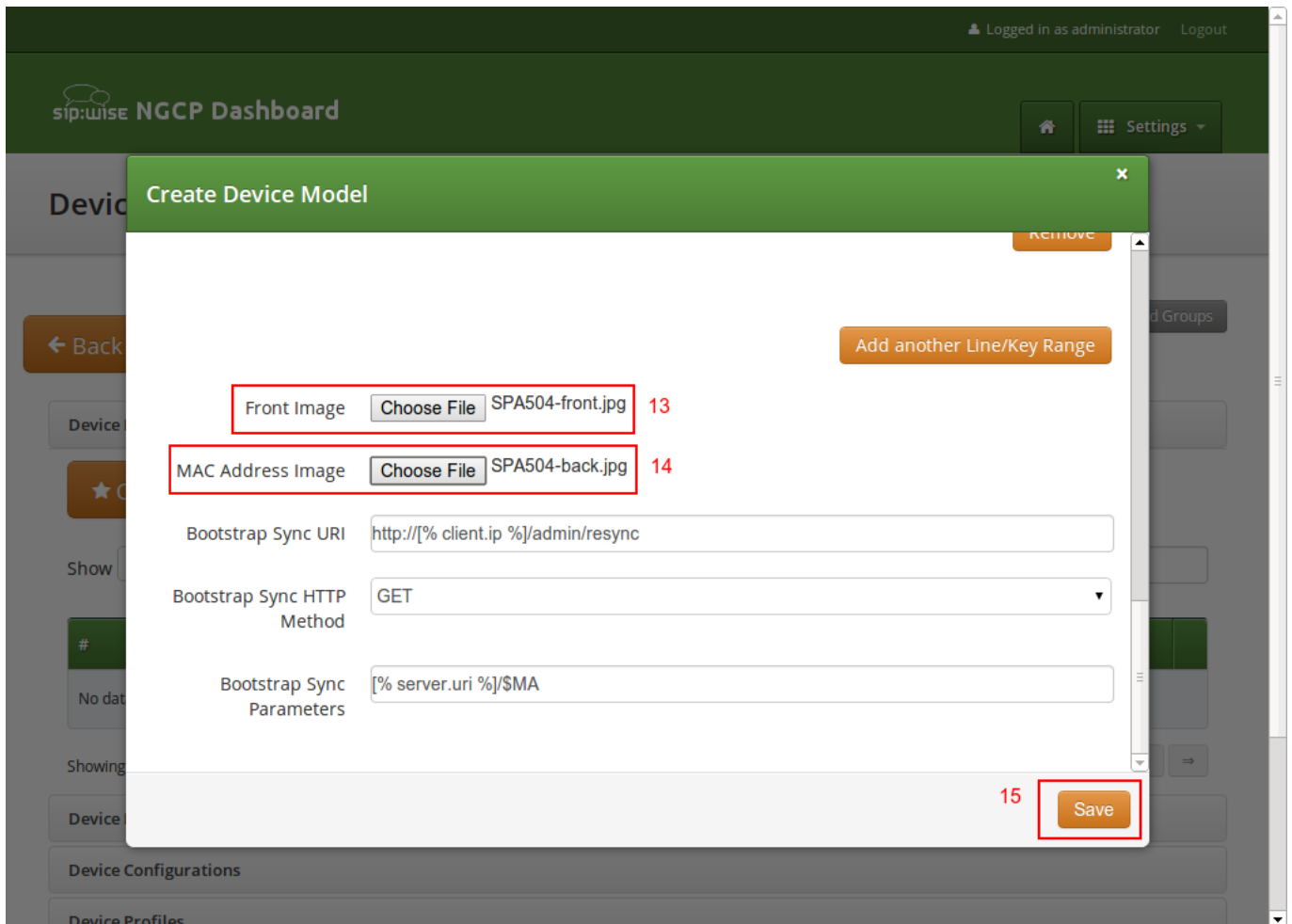


Figure 102: Create Device Model Part 4

### 16.1.1.2 Uploading Device Firmwares

A device model can optionally have one or more device firmware(s). Some devices like the Cisco SPA series don't support direct firmware updates from an arbitrary to the latest one, but need to go over specific firmware steps. In the device configuration discussed next, you can return the *next* supported firmware version, if the phone passes the current version in the firmware URL.

Since a stock phone purchased from any shop can have an arbitrary firmware version, we need to upload all firmwares needed to get from any old one to the latest one. In case of the Cisco SPA3x/SPA5x series, that would be the following versions, if the phone starts off with version 7.4.x:

- spa50x-30x-7-5-1a.bin
- spa50x-30x-7-5-2b.bin
- spa50x-30x-7-5-5.bin

So to get an SPA504G with a firmware version 7.4.x to the latest version 7.5.5, we need to upload each firmware file as follows.

Open the *Device Firmware* row in the *Device Management* section and press *Upload Device Firmware*.

Select the device model we're going to upload the firmware for, then specify the firmware version and choose the firmware file, then press *Save*.

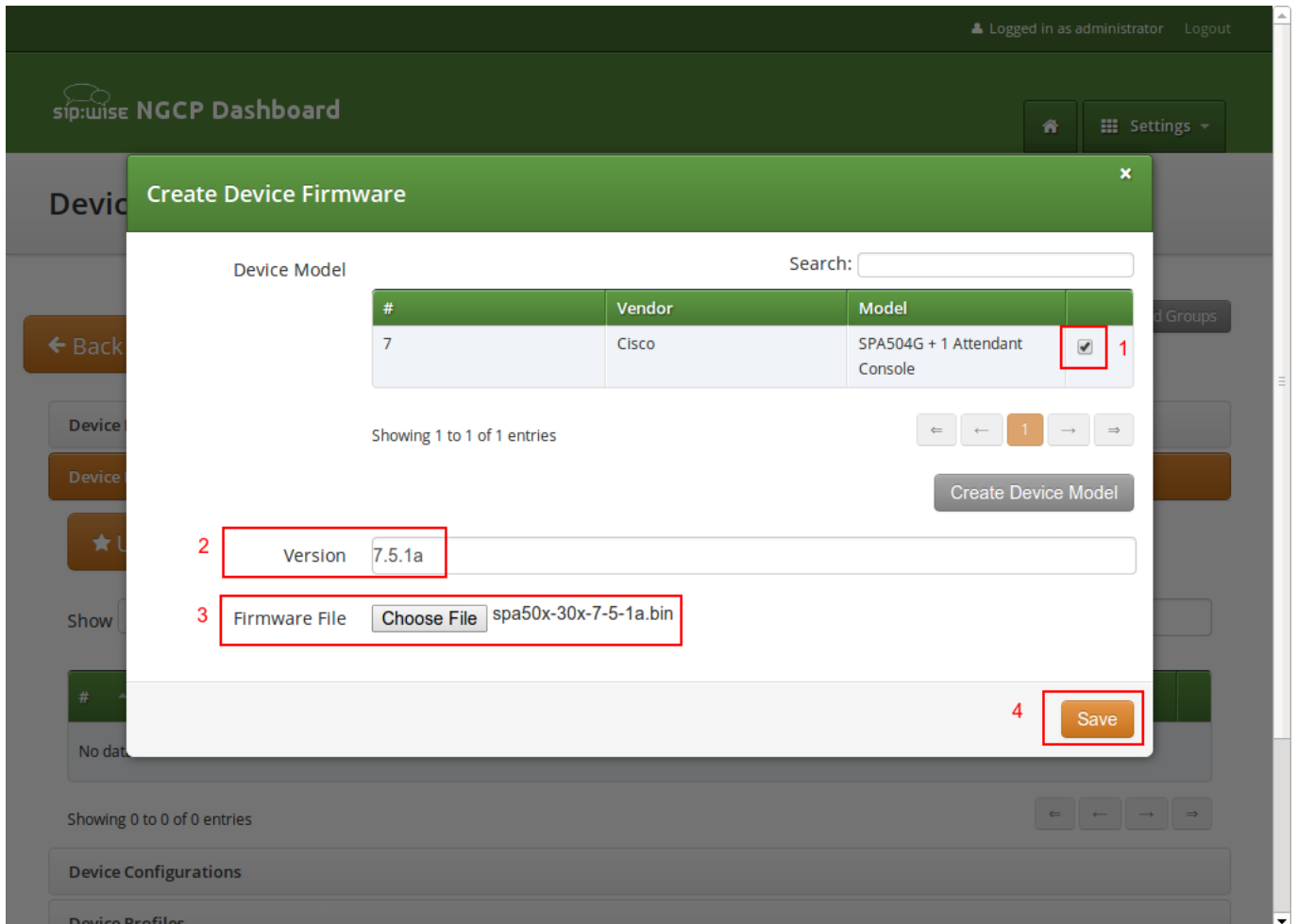


Figure 103: Upload Device Firmware

Repeat this step for every firmware in the list above (and any new firmware you want to support when it's available).

### 16.1.1.3 Creating Device Configurations

Each customer device needs a configuration file, which defines the URL to perform firmware updates, and most importantly, which defines the subscribers and features configured on each of the lines and keys. Since these settings are different for each physical phone at all the customers, the Cloud PBX module provides a template system to specify the configurations. That way, template variables can be used in the generic configuration, which are filled in by the system individually when a physical device fetches its configuration file.

To upload a configuration template, open the *Device Configuration* row and press *Create Device Configuration*.

Select the device model and specify a version number for this configuration (it is only for your reference to keep track of different

versions). For Cisco SPA phones, keep the *Content Type* field to `text/xml`, since the configuration content will be served to the phone as XML file.

For devices other than the Cisco SPA, you might set `text/plain` if the configuration file is plain text, or `application/octet-stream` if the configuration is compiled into some binary form.

Finally paste the configuration template into the *Content* area and press *Save*.

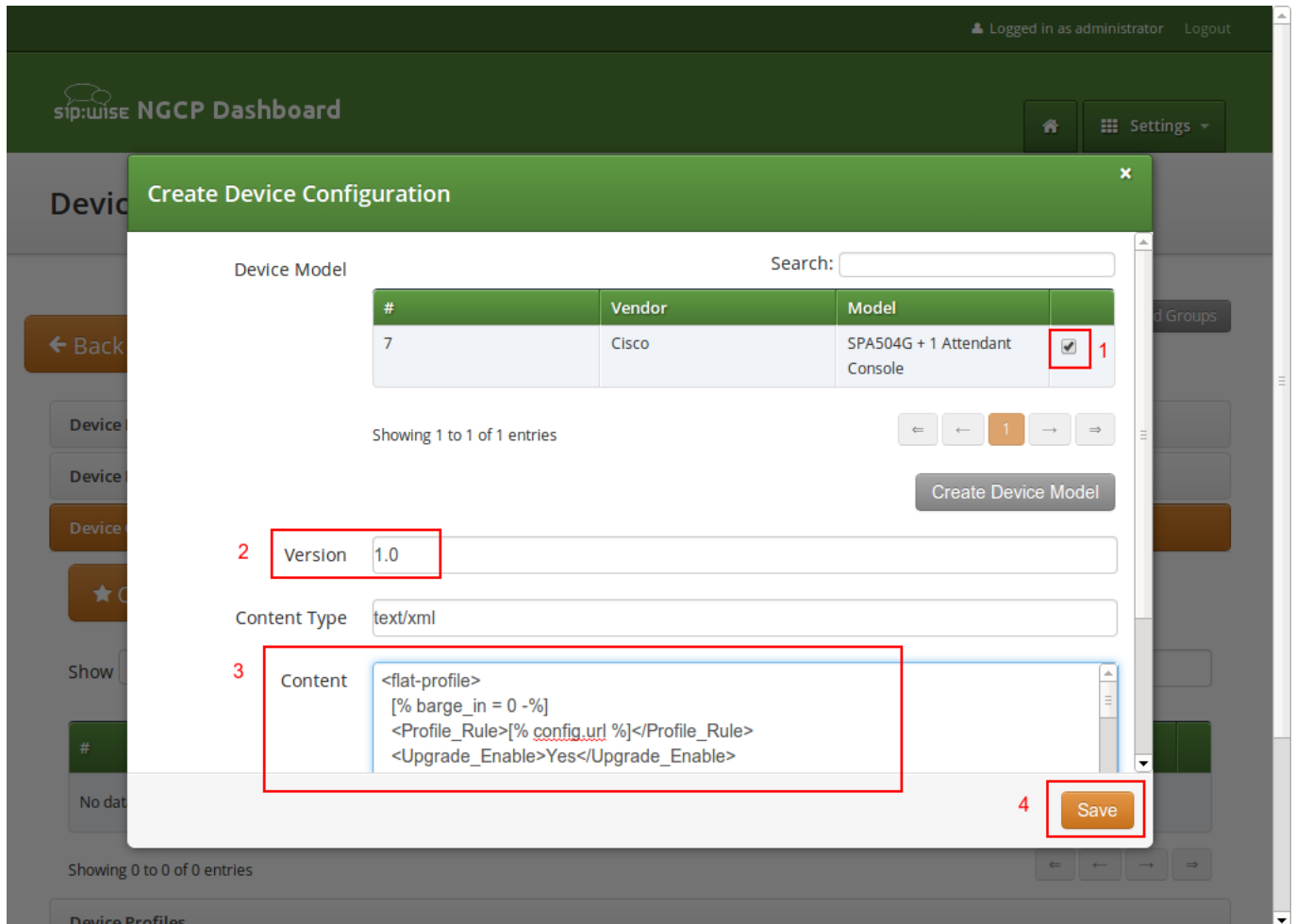


Figure 104: Upload Device Configuration

The templates for certified device models are provided by Sipwise, but you can also write your own. The following variables can be used in the template:

- `config.url`: The URL to the config file, including the device identifier (e.g. `http://sip.example.org:1444/device/autoprov/config/001122334455`).
- `firmware.maxversion`: The latest firmware version available on the system for the specific device.
- `firmware.baseurl`: The base URL to download firmwares (e.g. `http://sip.example.org:1444/device/autoprov/firmware`). To fetch the next newer firmware for a Cisco SPA, you can use the template line `[% firmware.baseurl %]/$MA/from/$SWVER/next`.



- `phone.stationname`: The name of the station (physical device) the customer specifies for this phone. Can be used to show on the display of the phone.
- `phone.lineranges`: An array of lines/keys as specified for the device model. Each entry in the array has the following keys:
  - `name`: The name of the line/key range as specified in the *Device Model* section (e.g. `Phone Keys`).
  - `num_lines`: The number of lines/keys in the line range (e.g. 4 in our `Phone Keys` example, or 32 in our `Attendant Console 1` example).
  - `lines`: An array of lines (e.g. subscriber definitions) for this line range. Each entry in the array has the following keys:
    - \* `keynum`: The index of the key in the line range, starting from 0 (e.g. `keynum` will be 3 for the 4th key of our `Phone Keys` range).
    - \* `rangenum`: The index of the line range, starting from 0. The order of line ranges is as you have specified them (e.g. `Phone Keys` was specified first, so it gets `rangenum 0`, `Auto Attendant 1` gets `rangenum 1`).
    - \* `type`: The type of the line/key, on of `private`, `shared` or `blf`.
    - \* `username`: The SIP username of the line.
    - \* `domain`: The SIP domain of the line.
    - \* `password`: The SIP password of the line.
    - \* `displayname`: The SIP Display Name of the line.

In the configuration template, you can adjust embedded variable references for the existing variables. If you need other specific variables, please request their development from Sipwise.

---

#### Tip

In order to change the provisioning base IP and port (default 1444), you have to access `/etc/ngcp-config/config.yml` and change the value `host` and `port` under the `autoprov.server` section.

---

#### 16.1.1.4 Creating Device Profiles

When the customer configures his own device, he doesn't select a *Device Model* directly, but a *Device Profile*. A device profile specifies which model is going to be used with which configuration version. This allows the operator to create new configuration files and assign them to a profile, while still keeping older configuration files for reference or roll-back scenarios. It also makes it possible to test new firmwares by creating a test device model with the new firmware and a specific configuration, without impacting any existing customer devices.

To create a *Device Profile* for our phone, open the *Device Profile* row in the *Device Management* section and press *Create Device Profile*.

Select the device configuration (which implicitly identifies a device model) and specify a *Profile Name*. This name is what the customer sees when he is selecting a device he wants to provision, so pick a descriptive name which clearly identifies a device. Press *Save* to create the profile.

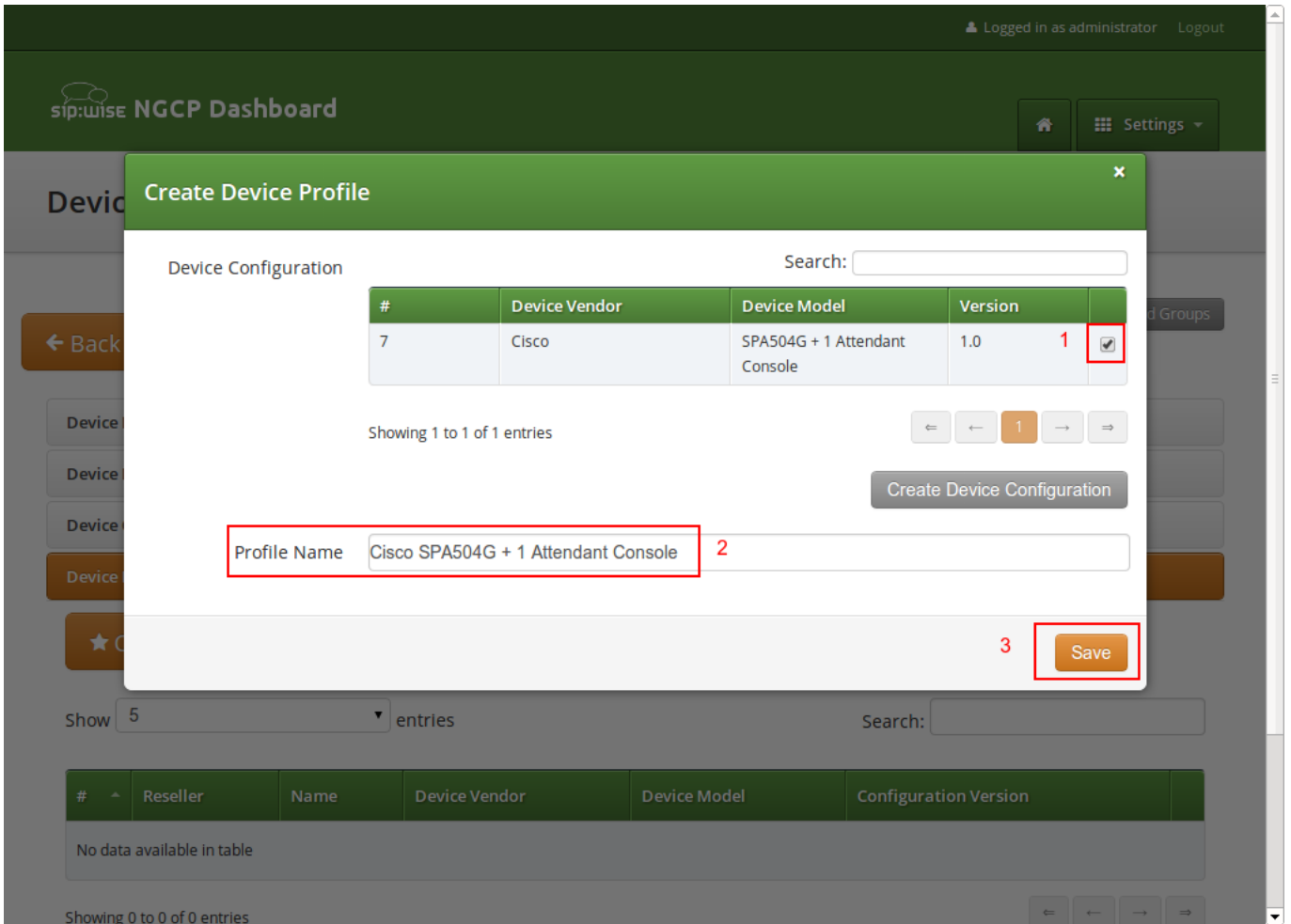


Figure 105: Create Device Profile

Repeat the steps as needed for every device you want to make available to customers.

### 16.1.2 Preparing PBX Rewrite Rules

In a PBX environment, the dial-plans usually looks different than for normal SIP subscribers. PBX subscribers should be able to directly dial internal extensions (e.g. 100) instead of the full number to reach another PBX subscriber in the same PBX segment. Therefore, we need to define specific *Rewrite Rules* to make this work.

The PBX dial plans are different from country to country. In the Central European area, you can directly dial an extension (e.g. 100), and if you want to dial an international number like 0049 1 23456, you have to dial a break-out digit first (e.g. 0), so the number to be dialed is 0 0049 1 23456. Other countries are used to other break-out codes (e.g. 9), which then results in 9 0049 1 23456. If you dial a national number like 01 23456, then the number to actually be dialed is 9 01 23456.

Since all numbers must be normalized to E.164 format via inbound rewrite rules, the rules need to be set up accordingly.

Let's assume that the break-out code for the example customers created below is 0, so we have to create a *Rewrite Rule Set* with the following rules.

### 16.1.2.1 Inbound Rewrite Rules for Caller

- **Match Pattern:** `^([1-9][0-9]+)$`
- **Replacement Pattern:** `${caller_cloud_pbx_base_cli}\1`
- **Description:** extension to e164
- **Direction:** Inbound
- **Field:** Caller

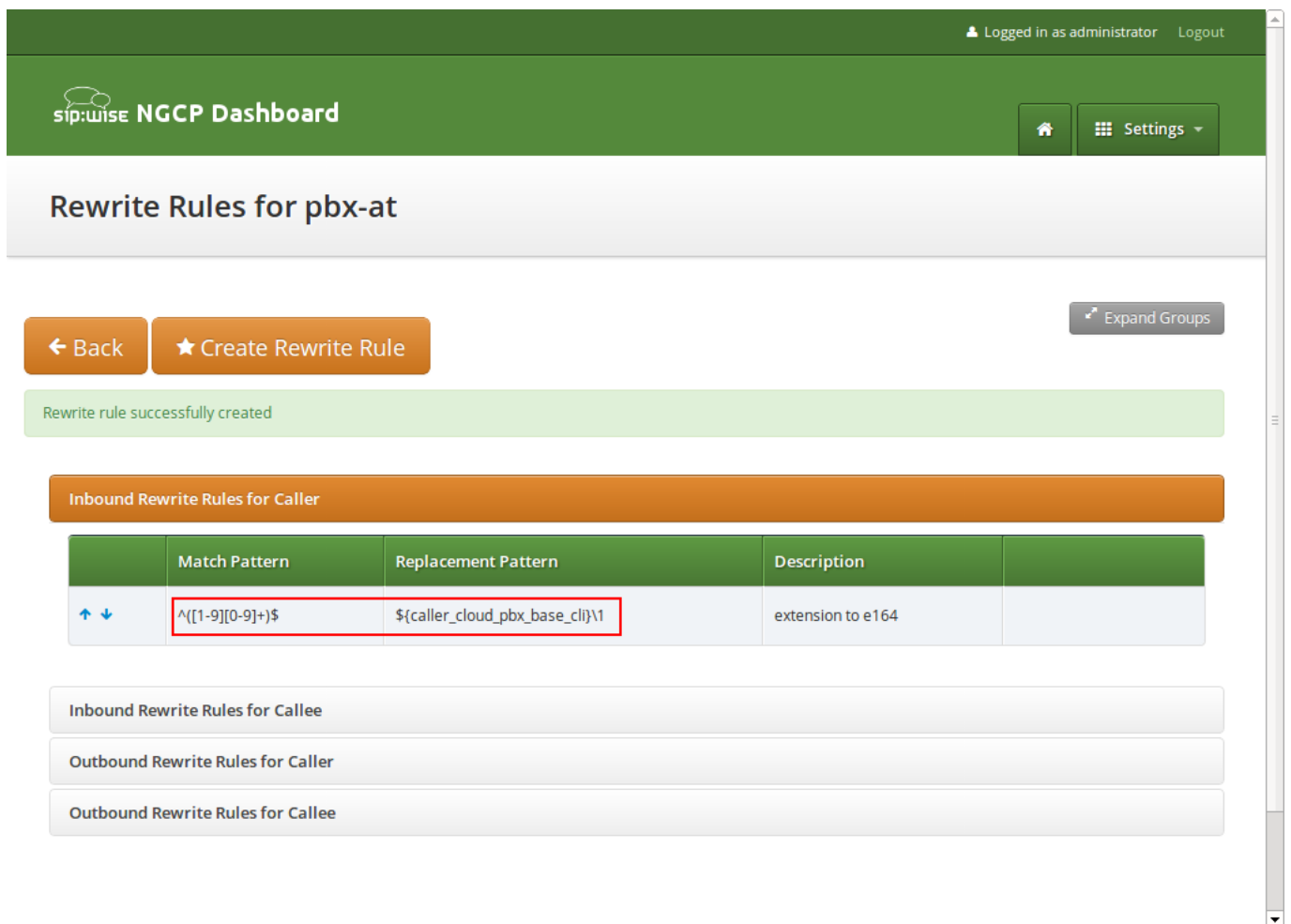


Figure 106: Inbound Rewrite Rule for Caller

### 16.1.2.2 Inbound Rewrite Rules for Callee

These rules are the most important ones, as they define which number formats the PBX subscribers can dial. For the break-out code of 0, the following rules are necessary e.g. for German dialplans to allow pbx internal extension dialing, local area calls without area codes, national calls with area code, and international calls with country codes.

PBX INTERNAL EXTENSION DIALIN

- **Match Pattern:** `^([1-9][0-9]+)$`
- **Replacement Pattern:** `${caller_cloud_pbx_base_cli}\1`
- **Description:** extension to e164
- **Direction:** Inbound
- **Field:** Callee

LOCAL DIALING WITHOUT AREA CODE (USE BREAK-OUT CODE 0)

- **Match Pattern:** `^0([1-9][0-9]+)$`
- **Replacement Pattern:** `${caller_cc}${caller_ac}\1`
- **Description:** local to e164
- **Direction:** Inbound
- **Field:** Callee

NATIONAL DIALING (USE BREAK-OUT CODE 0 AND PREFIX AREA CODE BY 0)

- **Match Pattern:** `^00([1-9][0-9]+)$`
- **Replacement Pattern:** `${caller_cc}\1`
- **Description:** national to e164
- **Direction:** Inbound
- **Field:** Callee

INTERNATIONAL DIALING (USE BREAK-OUT CODE 0 AND PREFIX COUNTRY CODE BY 00)

- **Match Pattern:** `^000([1-9][0-9]+)$`
- **Replacement Pattern:** `\1`
- **Description:** international to e164
- **Direction:** Inbound
- **Field:** Callee

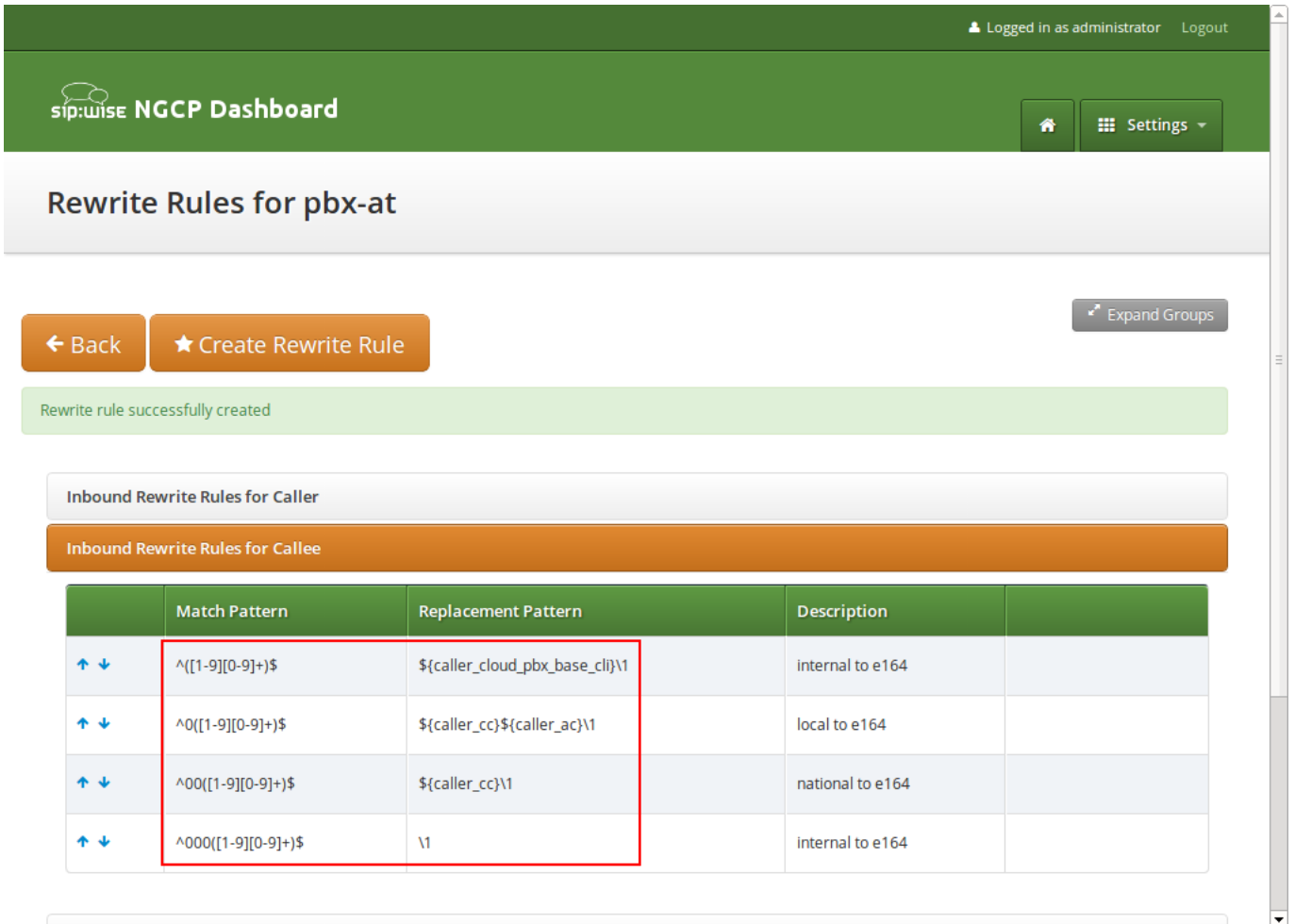


Figure 107: Inbound Rewrite Rule for Callee

### 16.1.2.3 Outbound Rewrite Rules for Caller

When a call goes to a PBX subscriber, it needs to be normalized in a way that it's call-back-able, which means that it needs to have the break-out code prefixed. We create a rule to show the calling number in international format including the break-out code. For PBX-internal calls, the caller name will be shown (this is handled by implicitly setting domain preferences accordingly, so you don't have to worry about that in rewrite rules).

ADDING A BREAK-OUT CODE (USE BREAK-OUT CODE 0 AND PREFIX COUNTRY CODE BY 00)

- **Match Pattern:** `^([1-9][0-9]+)$`
- **Replacement Pattern:** `000\1`
- **Description:** e164 to full international
- **Direction:** Outbound
- **Field:** Caller

## DISPLAYING THE EXTENSION IN THE CALLER NUMBER FOR PBX-INTERNAL CALLS

- **Match Pattern:** `^@{callee_cloud_pbx_account_cli_list}$`
- **Replacement Pattern:** `${caller_cloud_pbx_ext}`
- **Description:** e164 to full international
- **Direction:** Outbound
- **Field:** Caller

	Match Pattern	Replacement Pattern	Description	Enabled	
↑ ↓	<code>^@{callee_cloud_pbx_account_cli_list}\$</code>	<code>\${caller_cloud_pbx_ext}</code>	Intra-PBX to extension	yes	
↑ ↓	<code>^[[1-9][0-9]+\$</code>	0001	e164 to full international	yes	

Figure 108: Outbound Rewrite Rule for Caller

Create a new *Rewrite Rule Set* for each dial plan you'd like to support. You can later assign it to customer domains and even to subscribers, if a specific subscriber of a PBX customer would like to have his own dial plan.

### 16.1.3 Creating Customers and Pilot Subscribers

As with a normal SIP Account, you have to create a *Customer* contract per customer, and one *Subscriber*, which the customer can use to log into the web interface and manage his PBX environment.

#### 16.1.3.1 Creating a PBX Customer

Go to *Settings*→*Customers* and click *Create Customer*. We need a *Contact* for the customer, so press *Create Contact*.

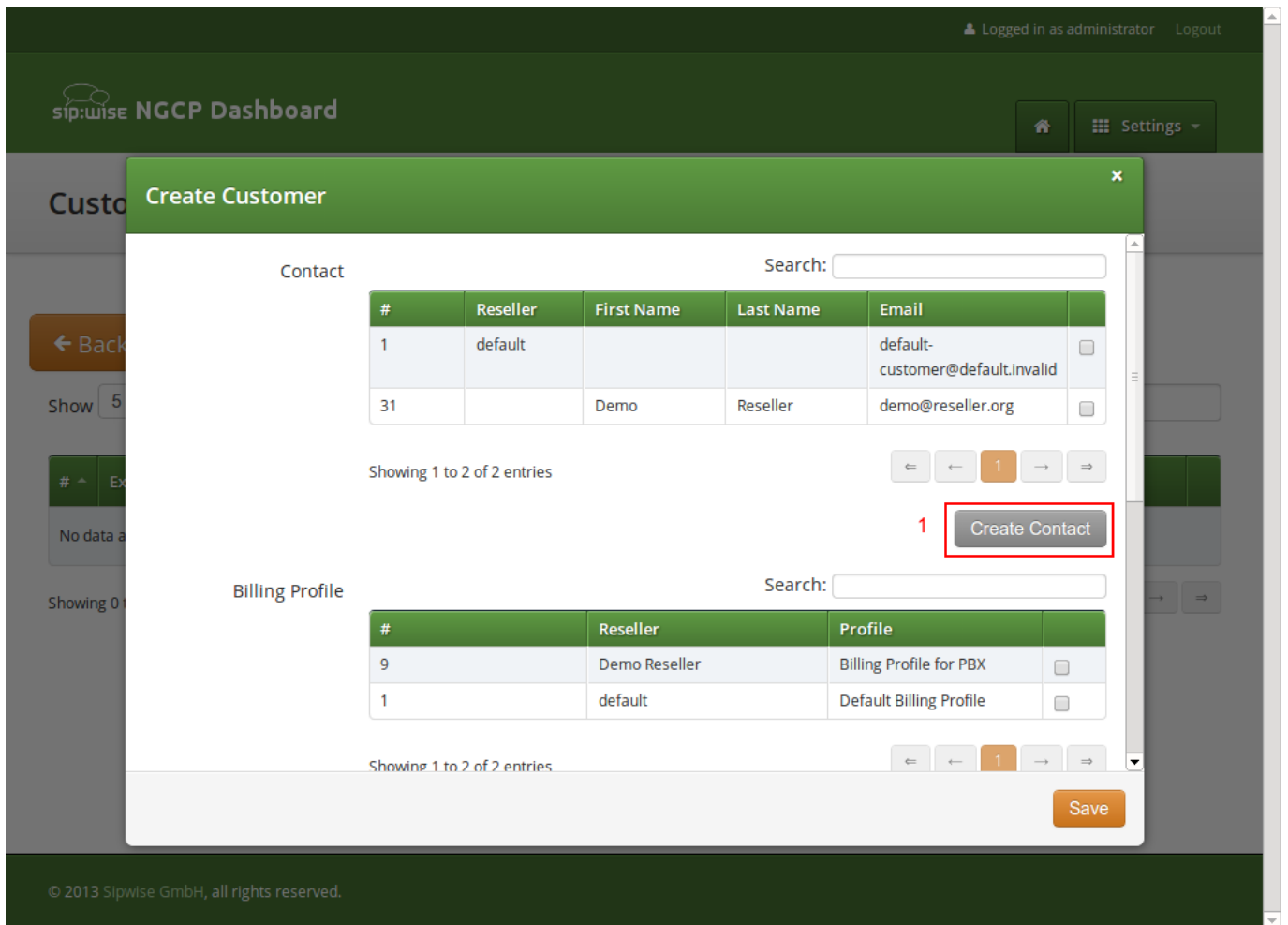


Figure 109: Create PBX Customer Part 1

Fill in the desired fields (you need to provide at least the *Email Address*) and press *Save*.

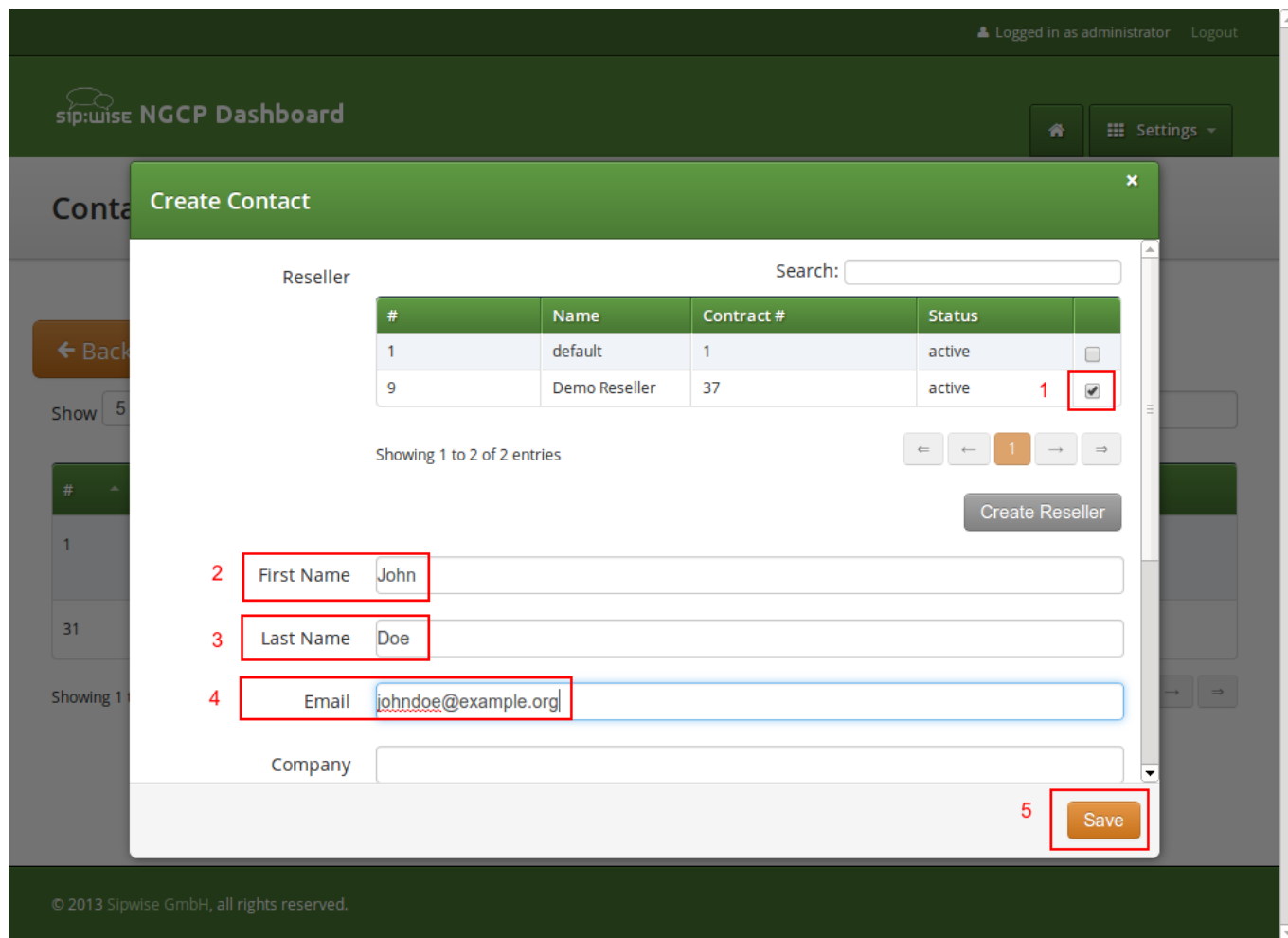


Figure 110: Create PBX Customer Contact

The new *Contact* will be automatically selected now. Also select a *Billing Profile* you want to use for this customer. If you don't have one defined yet, press *Create Billing Profile*, otherwise select the one you want to use.



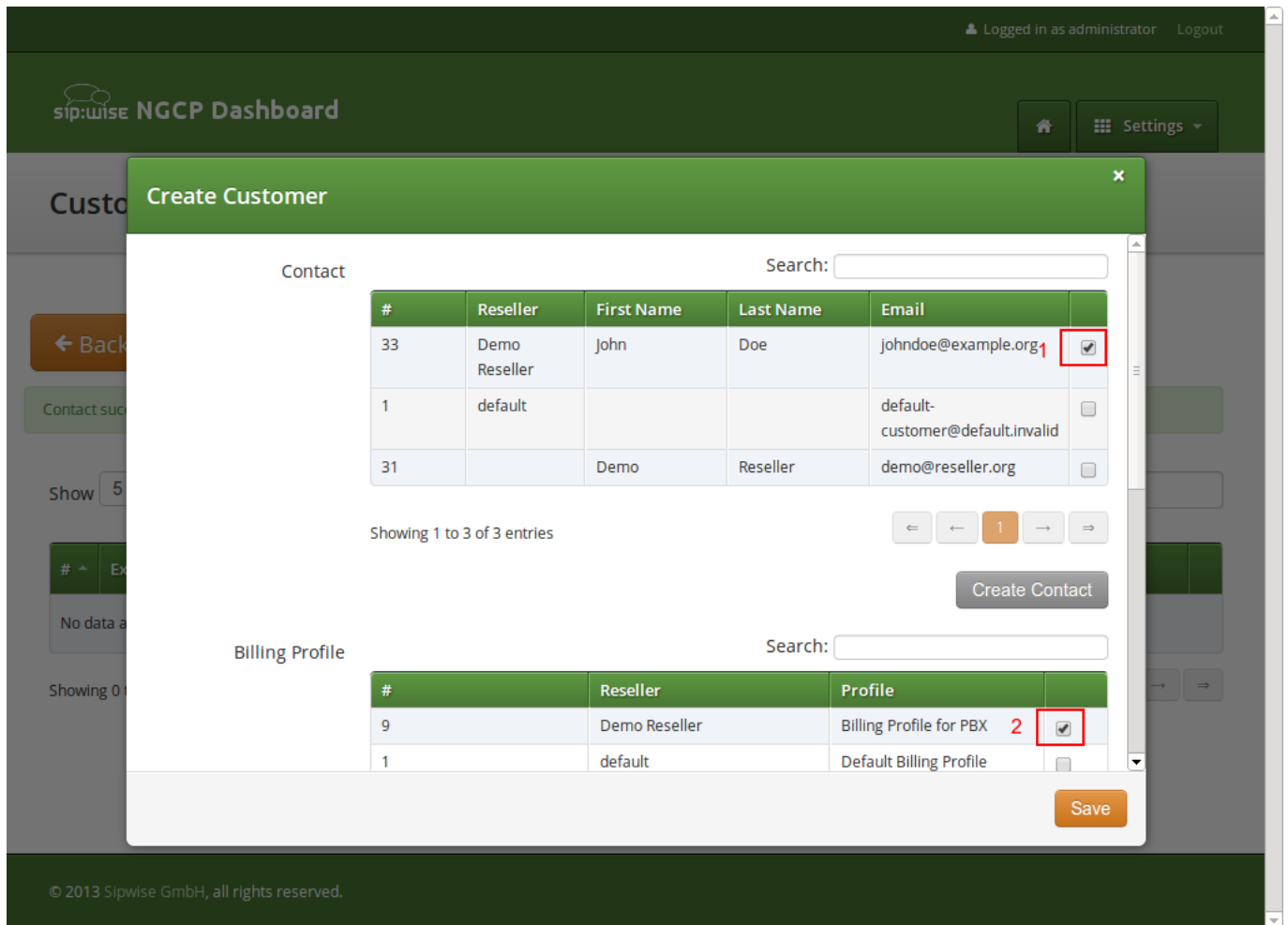


Figure 111: Create PBX Customer Part 2

Next, you need to select the *Product* for the PBX customer. Since it's going to be a PBX customer, select the product *Cloud PBX Account*.

Since PBX customers are supposed to manage their subscribers by themselves, they are able to create them via the web interface. To set an upper limit of subscribers a customer can create, define the value in the *Max Subscribers* field.



#### Important

As you will see later, both PBX subscribers and PBX groups are normal subscribers, so the value defined here limits the overall amount of subscribers **and** groups. A customer can create an unlimited amount of subscribers if you leave this field empty.

Press *Save* to create the customer.

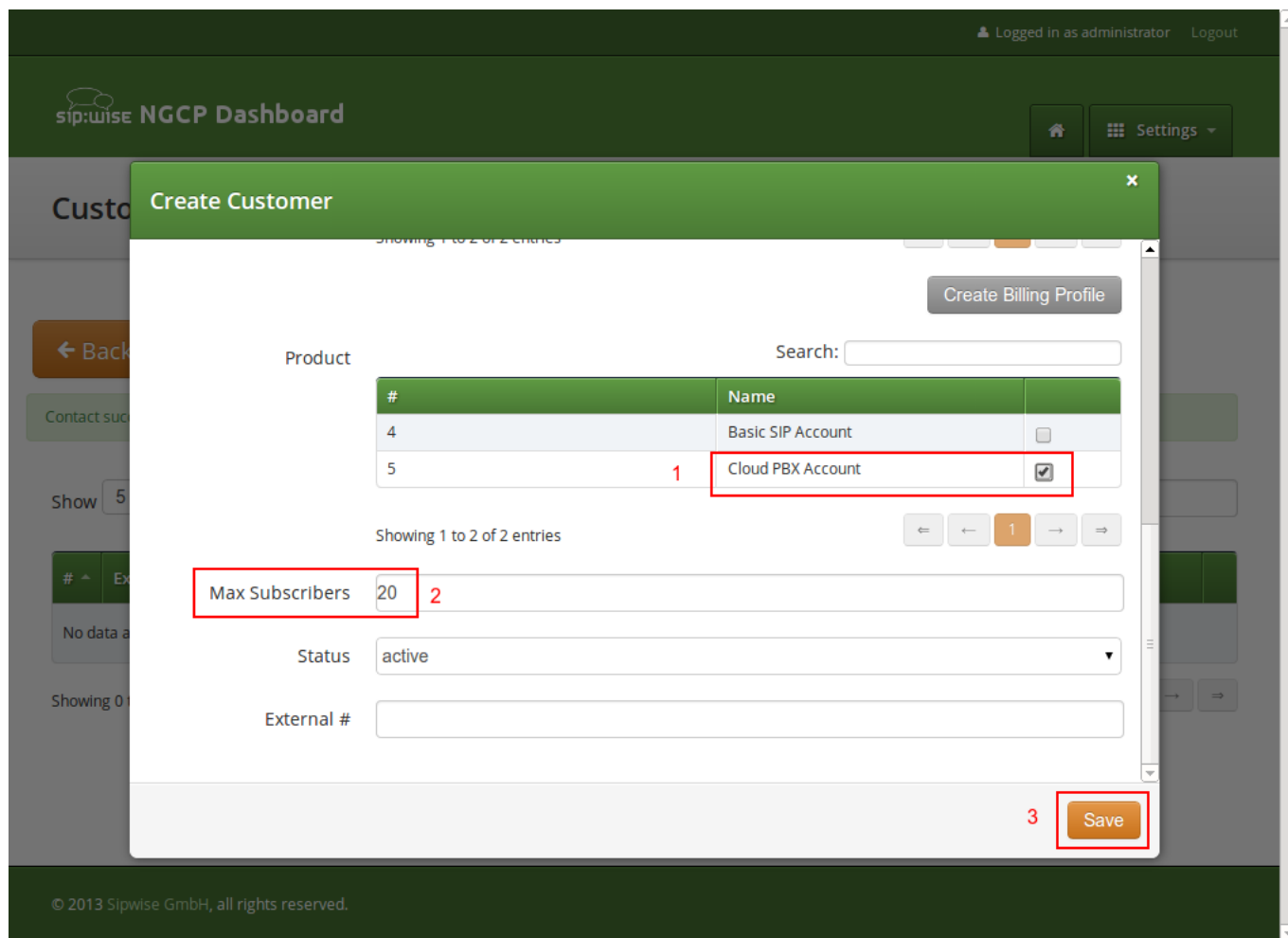


Figure 112: Create PBX Customer Part 3

### 16.1.3.2 Creating a PBX Pilot Subscriber

Once the customer is created, you need to create at least one *Subscriber* for the customer, so he can log into the web interface and manage the rest by himself.

Click the *Details* button on the newly created customer to enter the detailed view.

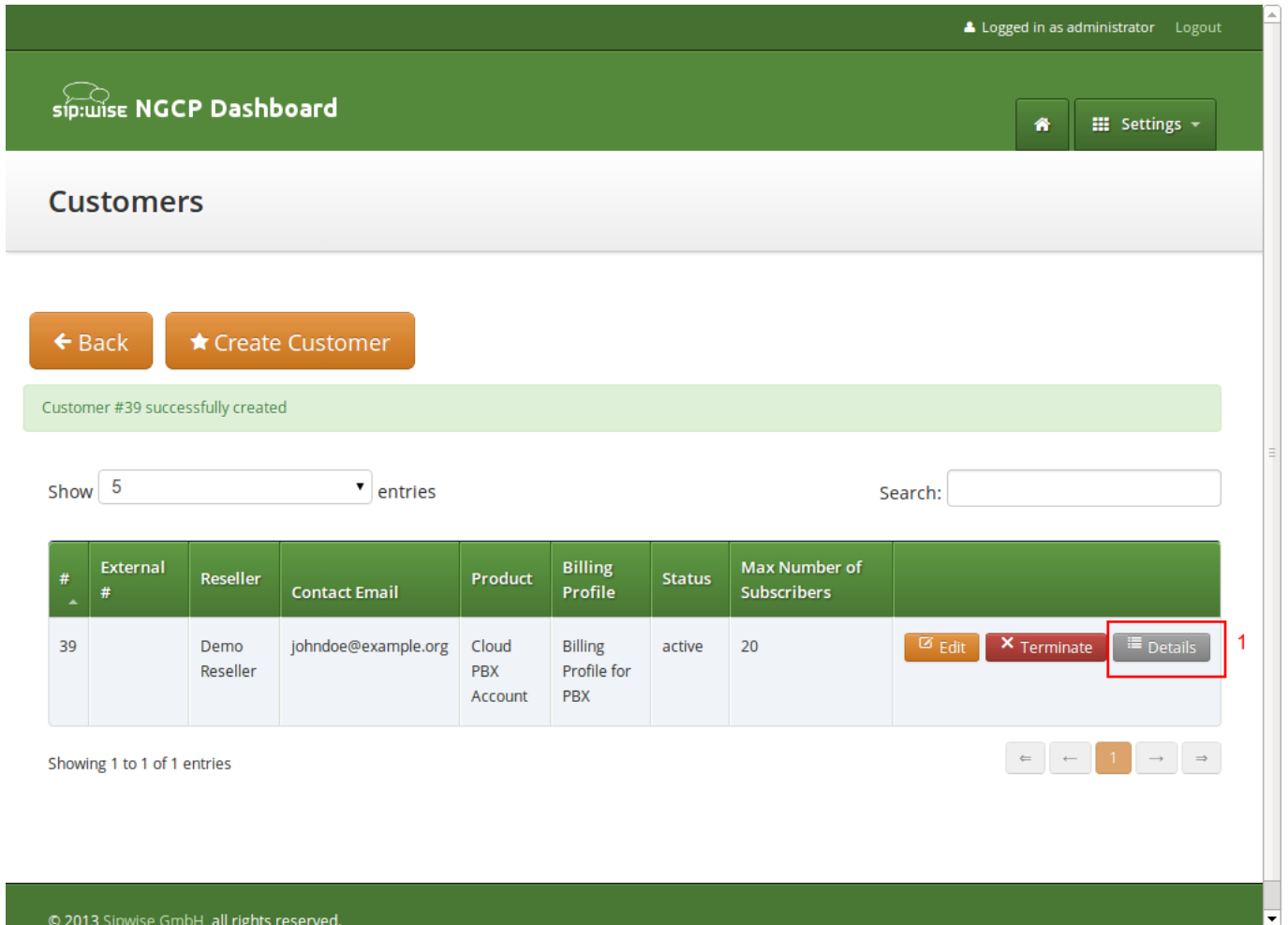


Figure 113: Go to Customer Details

To create the subscriber, open the *Subscribers* row and click *Create Subscriber*.

Logged in as administrator Logout

sip:wise NGCP Dashboard

Home Settings

## Customer Details for #39 (Cloud PBX Account)

Back Edit Expand Groups

Reseller

Contact Details

Billing Profiles

1 Subscribers

0 of maximum 20 subscribers (including PBX groups) created

2 ★ Create Subscriber

SIP URI	Primary Number	PBX Group	Registered Devices
---------	----------------	-----------	--------------------

Sound Sets

Contract Balance

Figure 114: Go to Create Subscriber

For your pilot subscriber, you need a SIP domain, a pilot number (the main number of the customer PBX), the web credentials for the customer to log into the web interfaces, and the SIP credentials to authenticate via a SIP device.

### Important



In a PBX environment, customers can create their own subscribers. As a consequence, each PBX customer should have its own SIP domain, in order to not collide with subscribers created by other customers. This is important because two customers are highly likely to create a subscriber (or group, which is also just a subscriber) called *office*. If they are in the same SIP domain, they'd both have the SIP URI *office@pbx.example.org*, which is not allowed, and the an end customer will probably not understand why *office@pbx.example.org* is already taken, because he (for obvious reasons, as it belongs to a different customer) will not see this subscriber in his subscribers list.

**Tip**

To handle one domain per customer, you should create a wild-card entry into your DNS server like `*.pbx.example.org`, which points to the IP address of `pbx.example.org`, so you can define SIP domains like `customer1.pbx.example.org` or `customer2.pbx.example.org` without having to create a new DNS entry for each of them. For proper secure access to the web interface and to the SIP and XMPP services, you should also obtain a SSL wild-card certificate for `*.pbx.example.org` to avoid certification warnings on customers' web browsers and SIP/XMPP clients.

So to create a new domain for the customer, click *Create Domain*.

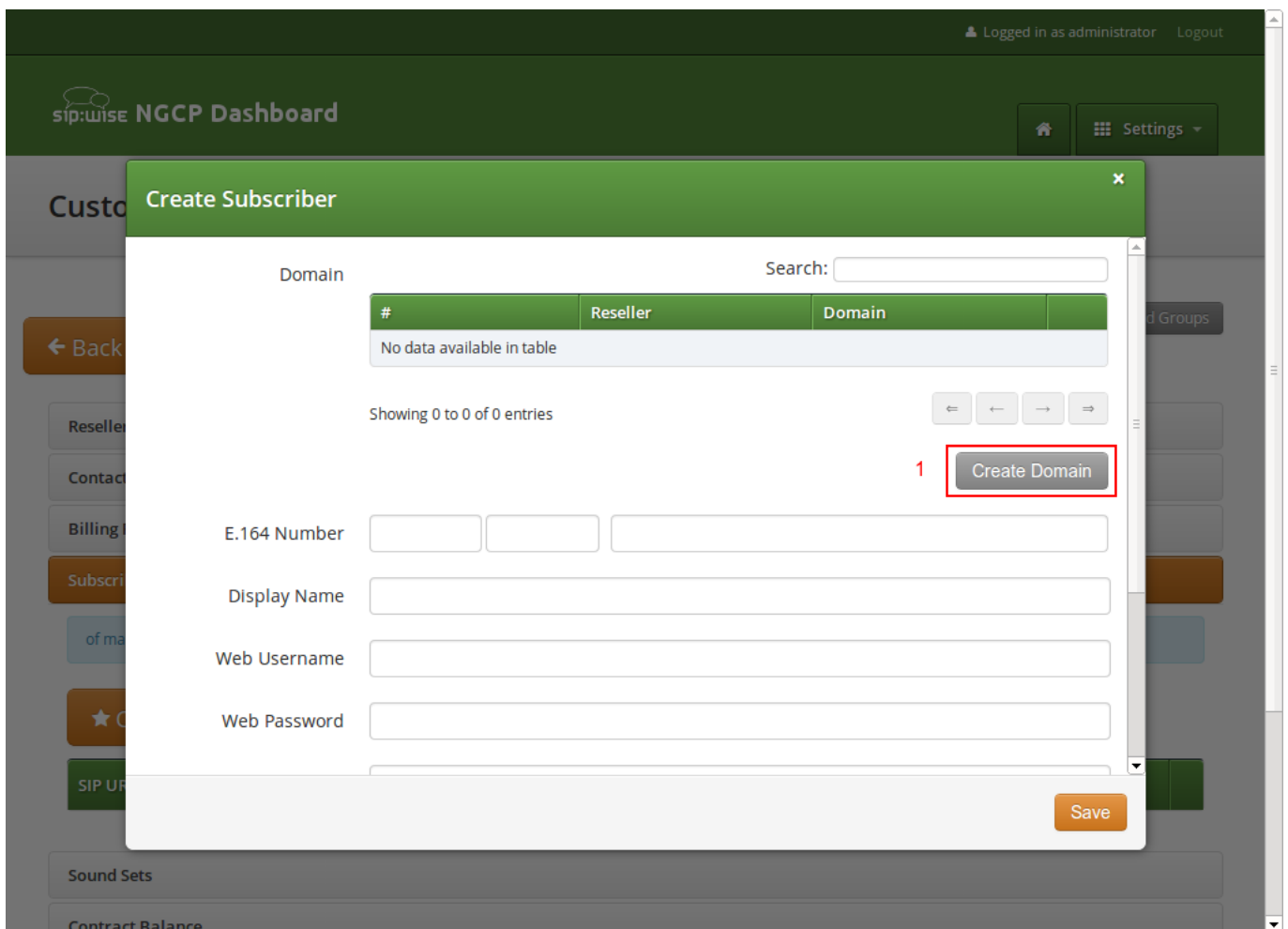


Figure 115: Go to Create Customer Domain

Specify the domain you want to create, and select the PBX *Rewrite Rule Set* which you created in Section 16.1.2, then click *Save*.

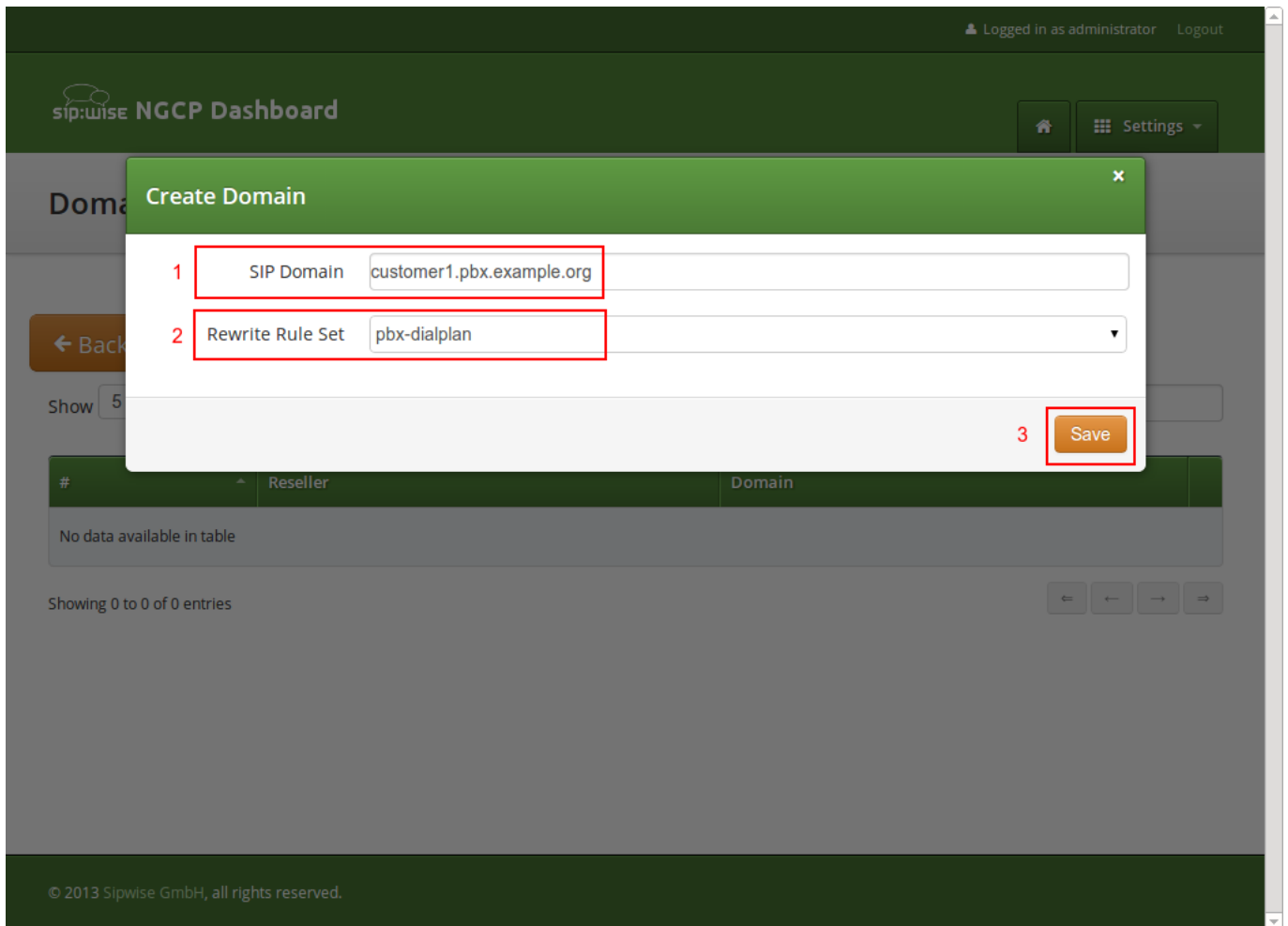


Figure 116: Create Customer Domain

Finish the subscriber creation by providing an E.164 number, which is going to be the base number for all other subscribers within this customer, the web username and password for the pilot subscriber to log into the web interface, and the sip username and password for a SIP device to connect to the PBX.

The parameters are as follows:

- **Domain:** The domain in which to create the pilot subscriber. *Each customer should get his own domain as described above to not collide with SIP usernames between customers.*
- **E.164 Number:** The primary number of the PBX. Calls to this number are routed to the pilot subscriber, and each subsequent subscriber created for this customer will use this number as its base number, suffixed by an individual extension. You can later assign alias numbers also for DID support.
- **Display Name:** This field is used on phones to identify subscribers by their real names instead of their number or extension. On outbound calls, the display name is signalled in the Display-Field of the From header, and it's used as a name in the XMPP contact lists.
- **Web Username:** The username for the subscriber to log into the customer self-care web interface. This is optional, if you don't

want a subscriber to have access to the web interface.

- **Web Password:** The password for the subscriber to log into the customer self-care web interface.
- **SIP Username:** The username for the subscriber to authenticate on the SIP and XMPP service. It is automatically used for devices, which are auto-provisioned via the *Device Management*, or can be used manually by subscribers to sign into the SIP and XMPP service with any arbitrary clients.
- **SIP Password:** The password for the subscriber to authenticate on the SIP and XMPP service.

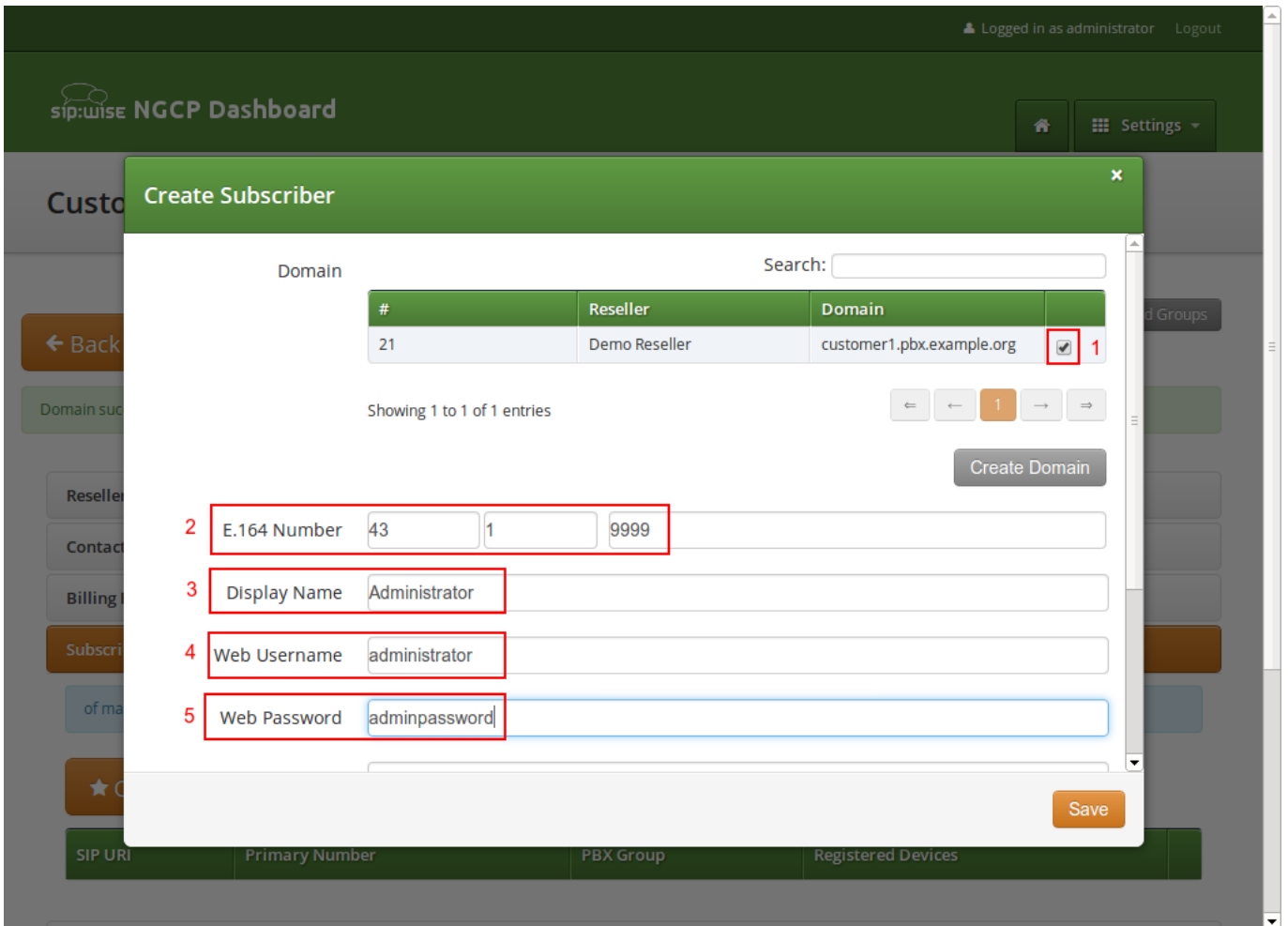


Figure 117: Create Pilot Subscriber Part 1

Customer Details for #39 (Cloud PBX Account)

**Create Subscriber**

E.164 Number

Display Name

Web Username

Web Password

1 SIP Username

2 SIP Password

Status

External ID

3

Figure 118: Create Pilot Subscriber Part 2

Once the subscriber is created, he can log into the customer self-care interface at <https://<your-ip>/login/subscriber> and manage his PBX, like creating other users and groups, assigning Devices to subscribers and configure the Auto Attendant and more.

As an administrator, you can also do this for the customer, and we will walk through the typical steps as an administrator to configure the different features.

Go to the *Customer Details* of the PBX customer you want to configure, e.g. by navigating to *Settings*→*Customers* and clicking the *Details* button of the customer you want to configure.

#### 16.1.4 Creating Regular PBX Subscribers

Since we already created a pilot subscriber, more settings now appear on the *Customer Details* view. The sections we are interested in for now are the *Subscribers* and *PBX Groups* sections.



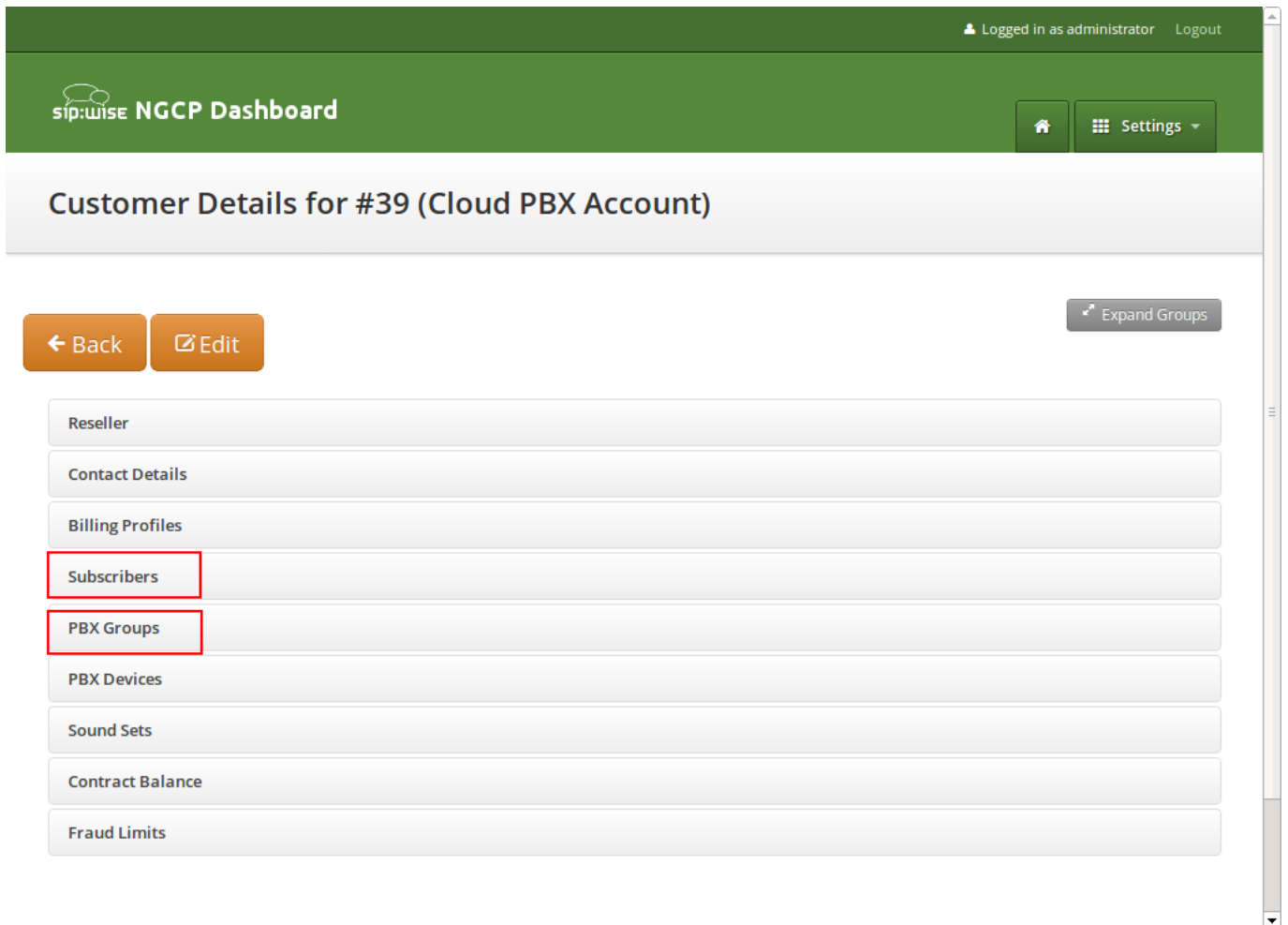


Figure 119: Subscribers and PBX Groups

To create another subscriber for the customer PBX, open the *Subscribers* row and click *Create Subscriber*.

Customer Details for #39 (Cloud PBX Account)

← Back   Edit   Expand Groups

Reseller

Contact Details

Billing Profiles

1 Subscribers

1 of maximum 20 subscribers (including PBX groups) created

2 ★ Create Subscriber

SIP URI	Primary Number	PBX Group	Registered Devices
administrator@customer1.pb.example.org	43 1 9999		

Figure 120: Create a Subscriber Extension

When creating another subscriber in the PBX after having the pilot subscriber, some fields are different now, because the *Domain* and *E.164 Number* are already pre-defined at the pilot subscriber level.

What you need to define for a new subscriber is the *Group* the subscriber is supposed to be in. We don't have a group yet, so create one by clicking *Create Group*.

A *PBX Group* has four settings:

- **Name:** The name of the group. This is used to identify a group when assigning it to subscribers on one hand, and also subscribers are pushed as server side contact lists to XMPP clients, where they are logically placed into their corresponding groups.
- **Extension:** The extension of the group, which is appended to the primary number of the pilot subscriber, so you can actually call the group from the outside. If our pilot subscriber number is 43 1 9999 and the extension is 100, you can reach the group from the outside by dialing 43 1 9999 100. Since PBX Groups are actually just normal subscribers in the system, you can assign *Alias Numbers* to it for DID later, e.g. 43 1 9998.
- **Hunting Policy:** If you call a group, then all members in this group are ringing based on the policy you choose. *Serial*

Ringling causes each of the subscribers to be tried one after another, until one of them picks up or all subscribers are tried. Parallel Ringing causes all subscribers in the group to be tried in parallel. Note that a subscriber can have a call-forward configured to some external number (e.g. his mobile phone), which will work as well.

- **Serial Hunting Timeout:** This value defines for how long to ring each member of a group in case of serial hunting until the next subscriber is being tried.

We will only fill in the *Name* and *Extension* for now, as the hunting policy can be changed later if needed. Click *Save* to create the group.

The screenshot shows the 'Create PBX Group' modal in the sip:wise NGCP Dashboard. The form contains the following fields:

- Name:** marketing (highlighted with a red box and labeled '1')
- Extension:** 100 (highlighted with a red box and labeled '2')
- Hunting Policy:** Serial Ringing (dropdown menu)
- Serial Hunting Timeout:** 10
- Save:** A button at the bottom right (highlighted with a red box and labeled '3')

Below the modal, a table shows the current state of subscribers:

SIP URI	Primary Number	PBX Group	Registered Devices
administrator@customer1.pbx.example.org	43 1 9999		

Figure 121: Create a PBX Group

Once the group is created and selected, fill out the rest of the form as needed. Instead of the *E.164 Number*, you can now only choose the *Extension*, which is appended to the primary number of the pilot subscriber and is then used as primary number for this particular subscribers. Again, if your pilot number is 43 1 9999 and you choose extension 101 here, the number of this subscriber is going to be 43 1 9999 101. Also, you can again later assign more alias numbers (e.g. 43 1 9997) to this subscriber for DID.

The rest of the fields is as usual, with *Display Name* defining the real name of the user, *Web Username* and *Web Password* allowing the subscriber to log into the customer self-care interface, and the *SIP Username* and *SIP Password* to allow signing into

the SIP and XMPP services.

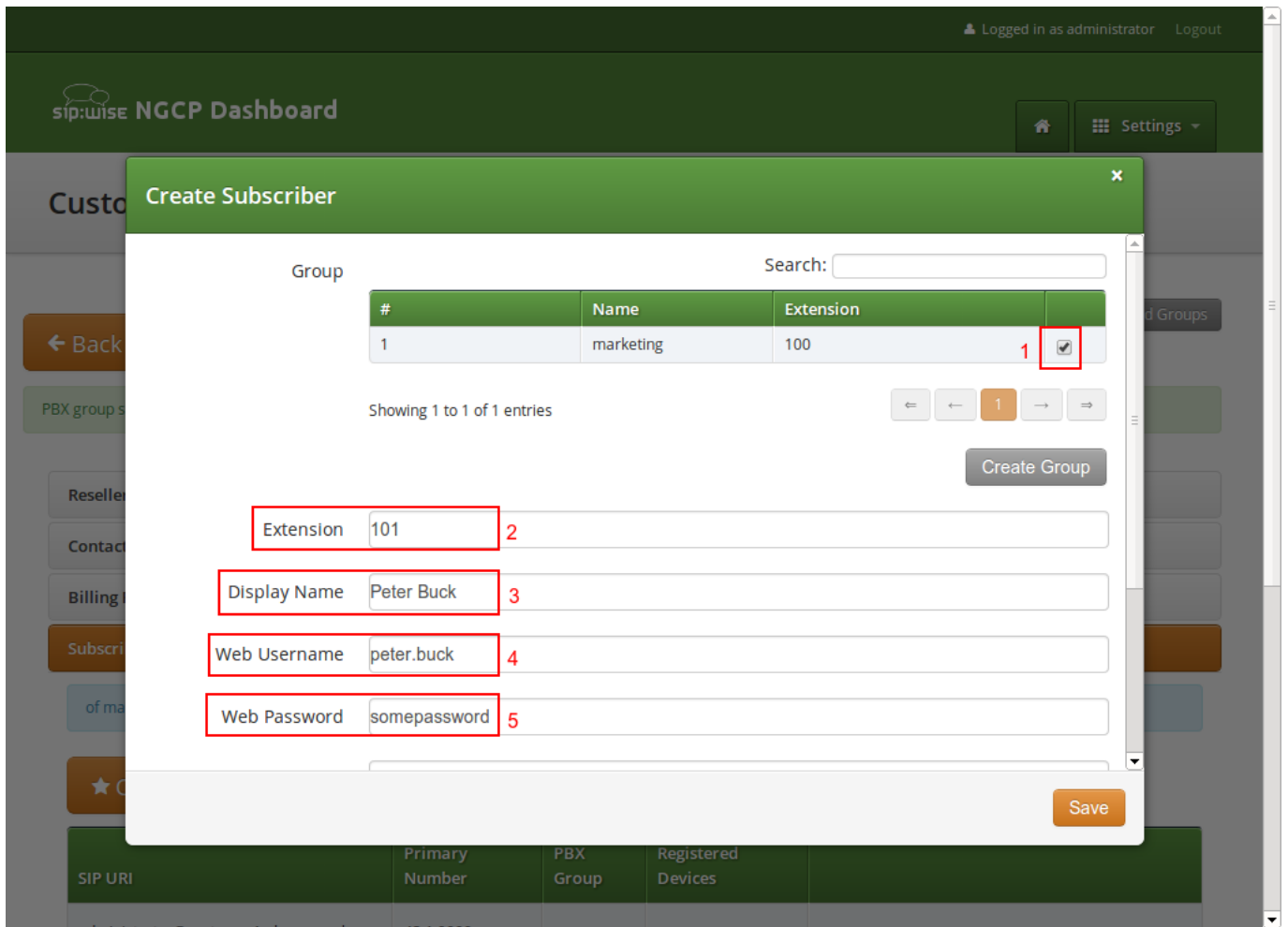


Figure 122: Finish PBX Subscriber Creation Part 1

Click Save to create the subscriber.

Customer Details for #39 (Cloud PBX Account)

### Create Subscriber

Extension: 101

Display Name: Peter Buck

Web Username: peter.buck

Web Password: somepassword

SIP Username: peter.buck **1**

SIP Password: somepassword **2**

Status: active

External ID:

**3** Save

Figure 123: Finish PBX Subscriber Creation Part 2

Repeat the steps to create all the subscribers and groups as needed. An example of a small company configuration in terms of subscribers and groups might look like this:

Reseller

Contact Details

Billing Profiles

Subscribers

7 of maximum 20 subscribers (including PBX groups) created

★ Create Subscriber

SIP URI	Primary Number	PBX Group	Registered Devices
administrator@customer1.pbx.example.org	43 1 9999		
peter.buck@customer1.pbx.example.org	43 1 9999101	marketing	
michelle.miller@customer1.pbx.example.org	43 1 9999102	marketing	
frank.fowler@customer1.pbx.example.org	43 1 9999201	development	
deborah.dane@customer1.pbx.example.org	43 1 9999202	development	

PBX Groups

PBX Devices

Sound Sets

Figure 124: Example of Subscribers List

**Tip**

The subscribers can be reached via 3 different ways. First, you can call them by their SIP URIs (e.g. by dialing `frank.fowler@customer1.pbx.example.org`) from both inside and outside the PBX. Second, you can dial by the full number (e.g. `43 1 9999 201`; depending on your rewrite rules, you might need to add a leading `\+` or `00` or leave out the country code when dialing from the outside, and adding a `0` as break-out digit when dialing from the inside) from both inside and outside the PBX. Third, you can dial just the extension (e.g. `201`) from inside the PBX. If the subscriber also has an alias number assigned, you can dial that number also, according to your dial-plan in the rewrite rules.

**16.1.5 Assigning Subscribers to Devices**

Basically you can register any SIP phone to the system using the SIP credentials of your subscribers. However, the platform supports *Device Provisioning* of certain vendors and models, as described in Section 16.1.1.

To configure a physical device, open the *PBX Devices* row in the *Customer Details* view and click *Create Device*.

You have to set three general parameters for your new device, which are:

- **Device Profile:** The actual device profile you want to use. This has been pre-configured in the *Device Management* by the administrator or reseller, and the customer can choose from the list of profiles (which is a combination of an actual device plus its corresponding configuration).
- **MAC Address/Identifier:** The MAC address of the phone to be added. The information can usually either be found on the back of the phone, or in the phone menu itself.
- **Station Name:** Since you can (depending on the actual device) configure more lines on a phone, you can give it a station name, like `Reception` or the name of the owner of the device.

In addition to that information, you can configure the lines (subscribers) you want to use on which key, and the mode of operation (e.g. if it's a normal private phone line, or if you want to monitor another subscriber using BLF, or if you want it to act as shared line using SLA).

For example, a *Cisco SPA504G* has 4 keys you can use for private and shared lines as well as BLF on the phone itself, and in our example we have an *Attendant Console* attached to it as well, so you have 32 more keys for BLF.

The settings per key are as follows:

- **Subscriber:** The subscriber to use (for private/shared lines) or to monitor (for BLF).
- **Line/Key:** The key where to configure this subscriber to.
- **Line/Key Type:** The mode of operation for this key, with the following options (depending on which options are enabled in the *Device Model* configuration for this device):
  - **Private Line:** Use the subscriber as a regular SIP phone line. This means that the phone will register the subscriber, and you can place and receive phone calls with/for this subscriber.
  - **Shared Line:** The subscriber is also registered on the system and you can place and receive calls. If another phone has the same subscriber also configured as shared line, both phones will ring on incoming calls, and you can pick the call up on either of them. You cannot place a call with this subscriber though if the line is already in use by another subscriber. However, you can "steal" a running call by pressing the key where the shared line is configured to barge into a running call. The other party (the other phone where the shared line is configured too) will then be removed from the call (but can steal the call back the same way).
  - **BLF Key:** The *Busy Lamp Field* monitors the call state of another subscriber and provides three different functionalities, depending on the actual state:
    - \* **Speed Dial:** If the monitored subscriber is on-hook, the user can press the button and directly call the monitored subscriber.
    - \* **Call Pickup:** If the monitored subscriber is ringing, the user can press the button to pick up the call on his own phone.
    - \* **State Indication:** If the monitored subscriber is on the phone, the key is indicating that the monitored subscriber is currently busy.

In our example, we will first configure a private line on the first key, and BLF for another subscriber on the second key.

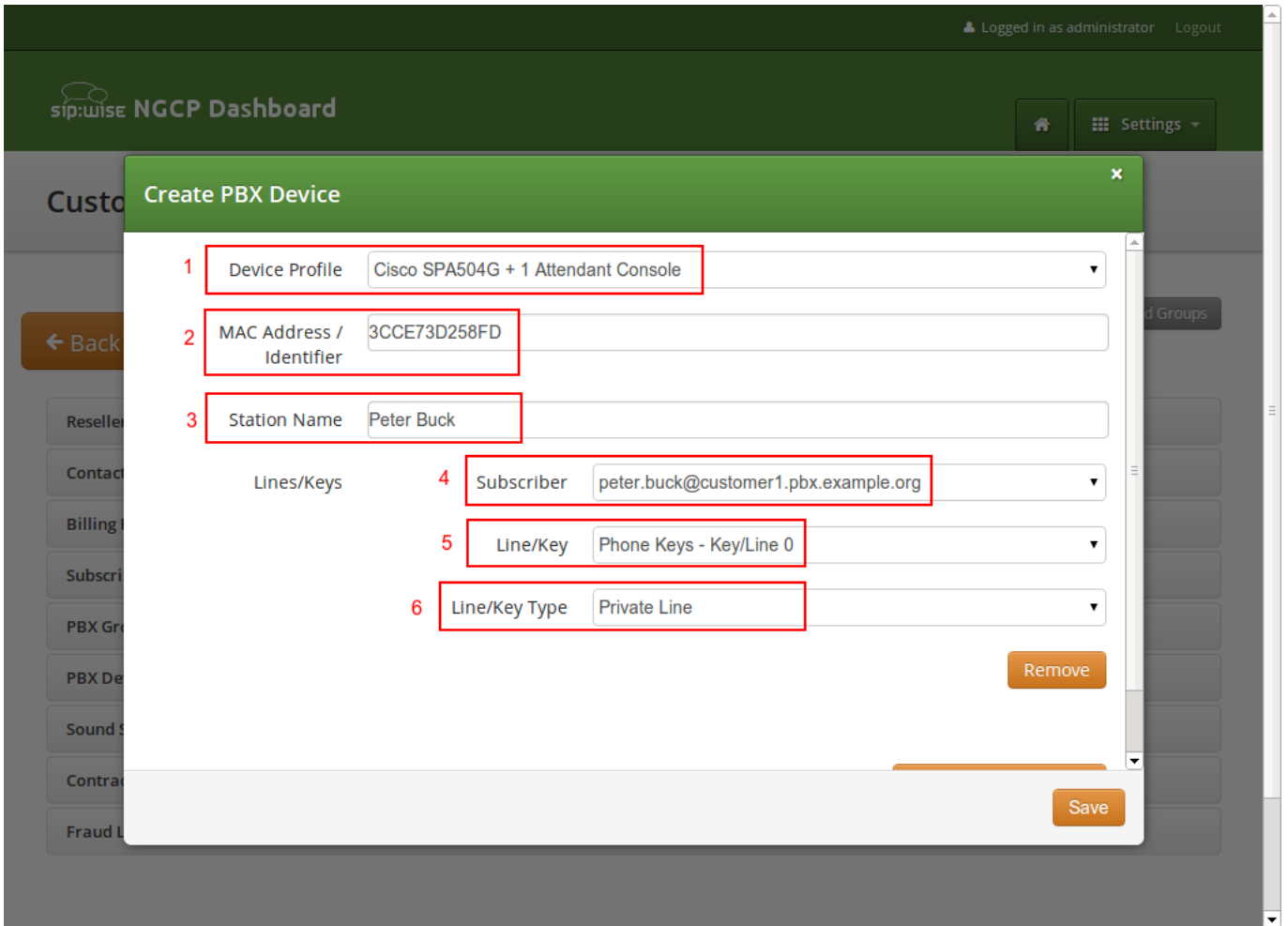
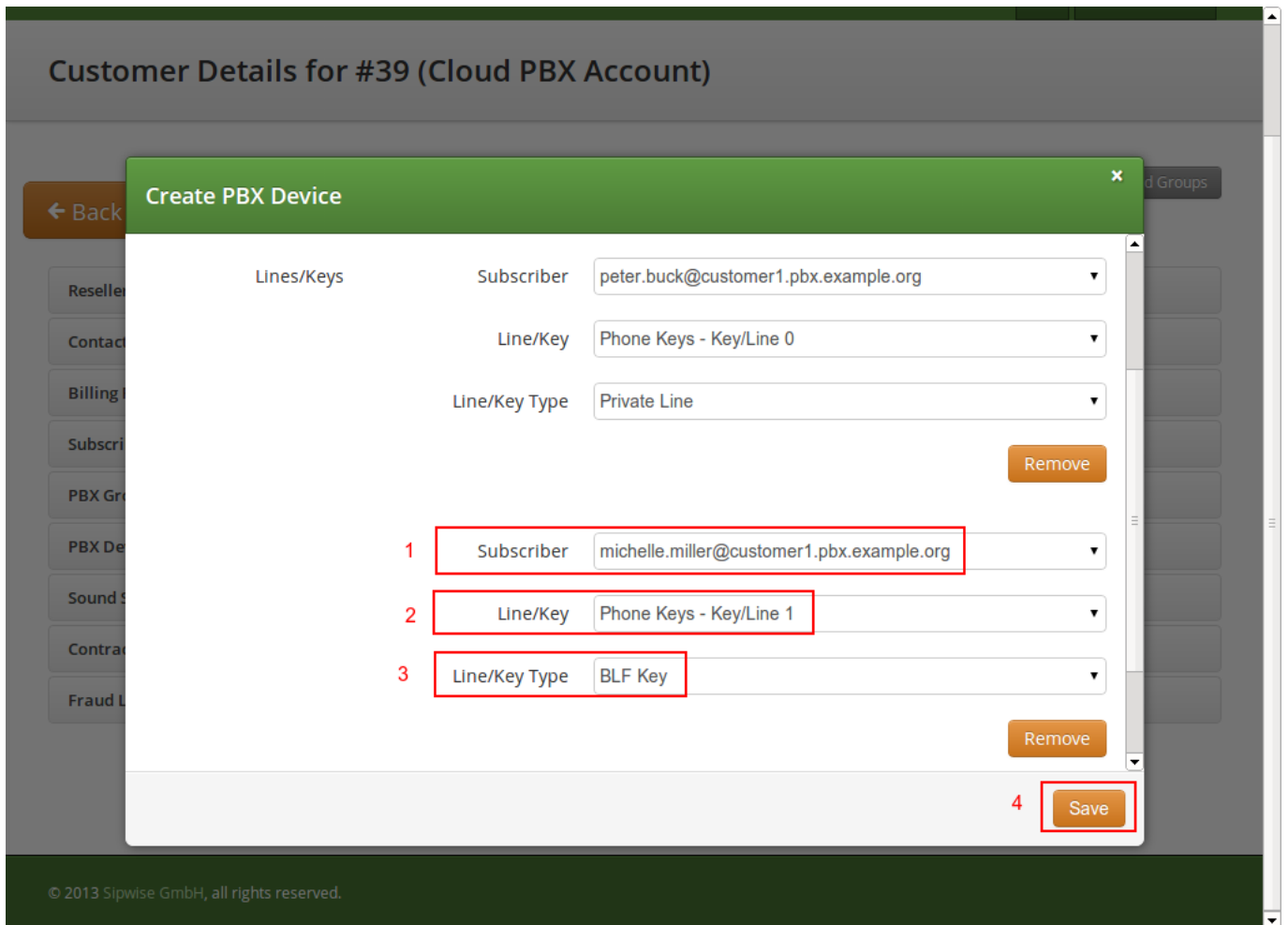


Figure 125: Configuring a PBX Device Part 1

This configures the general options plus the first key. To configure the second key, click *Add another Line/Key* and fill out the second line config accordingly. Click *Save* to save your PBX device configuration.





Customer Details for #39 (Cloud PBX Account)

← Back

Create PBX Device

Lines/Keys

Subscriber peter.buck@customer1.pbx.example.org

Line/Key Phone Keys - Key/Line 0

Line/Key Type Private Line

Remove

1 Subscriber michelle.miller@customer1.pbx.example.org

2 Line/Key Phone Keys - Key/Line 1

3 Line/Key Type BLF Key

Remove

4 Save

© 2013 Sipwise GmbH, all rights reserved.

Figure 126: Configuring a PBX Device Part 2

Once the PBX device is saved, you will see it in the list of *PBX Devices*.

#### 16.1.5.1 Synchronizing a PBX Device for Initial Usage

Since a stock device obtained from an arbitrary distributor doesn't know anything about your system, it can't fetch its configuration from there. For that to work, you need to push the URL of where the phone can get the configuration to the phone once.

In order to do so, click the *Sync Device* button on the device you want to configure for the very first time.

Contact Details


Billing Profiles

Subscribers

PBX Groups

PBX Devices

★ Create PBX Device

	Station Name	Subscriber	MAC Address / Identifier	Device Profile	
	Peter Buck	Phone Keys/0: private peter.buck@customer1.pbx.example.org  Phone Keys/1: blf michelle.miller@customer1.pbx.example.org	3cce73d258fd	Cisco SPA504G + 1 Attendant Console	<span style="background-color: #a00; color: white; padding: 2px 5px; border: 1px solid #a00;">✕ Delete</span> <span style="background-color: #d95319; color: white; padding: 2px 5px; border: 1px solid #d95319;">✎ Edit</span> <span style="background-color: #d95319; color: white; padding: 2px 5px; border: 2px solid #d95319;">☑ Sync Device</span> <span style="color: red; font-weight: bold; font-size: 1.2em; margin-left: 5px;">1</span>

Sound Sets

Contract Balance


Fraud Limits

Figure 127: Go to Sync Device



**Important**

As you will see in the next step, you need the actual IP address of the phone to push the provisioning URL onto it. That implies that you need access to the phone to get the IP, and that your browser is in the same network as the phone in order to be able to connect to it, in case the phone is behind NAT.

Enter the IP Address of the phone (on Cisco SPAs, press *Settings* , where *Settings* is the paper sheet symbol, and note down the *Current IP* setting), then click *Push Provisioning URL*.

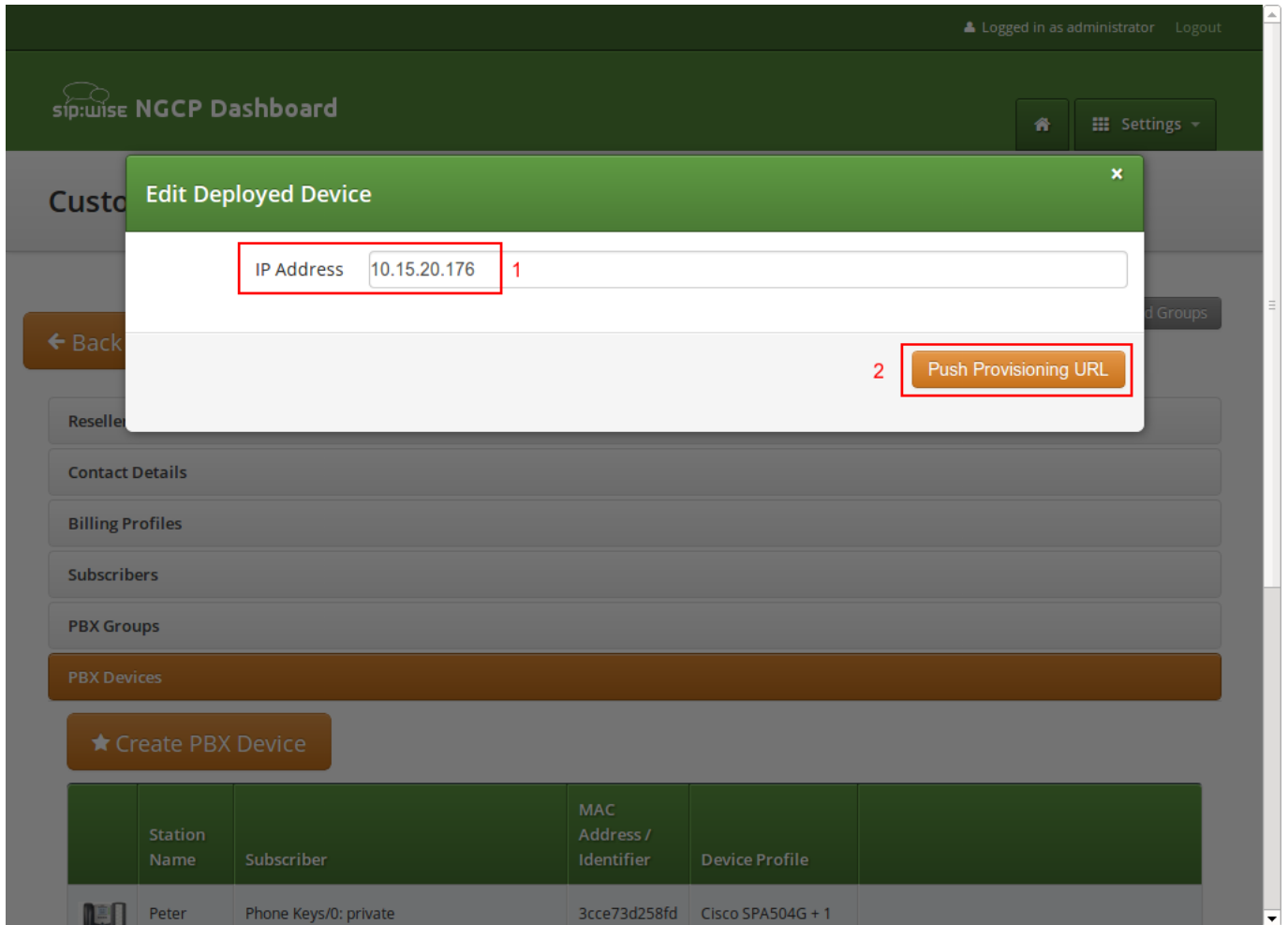
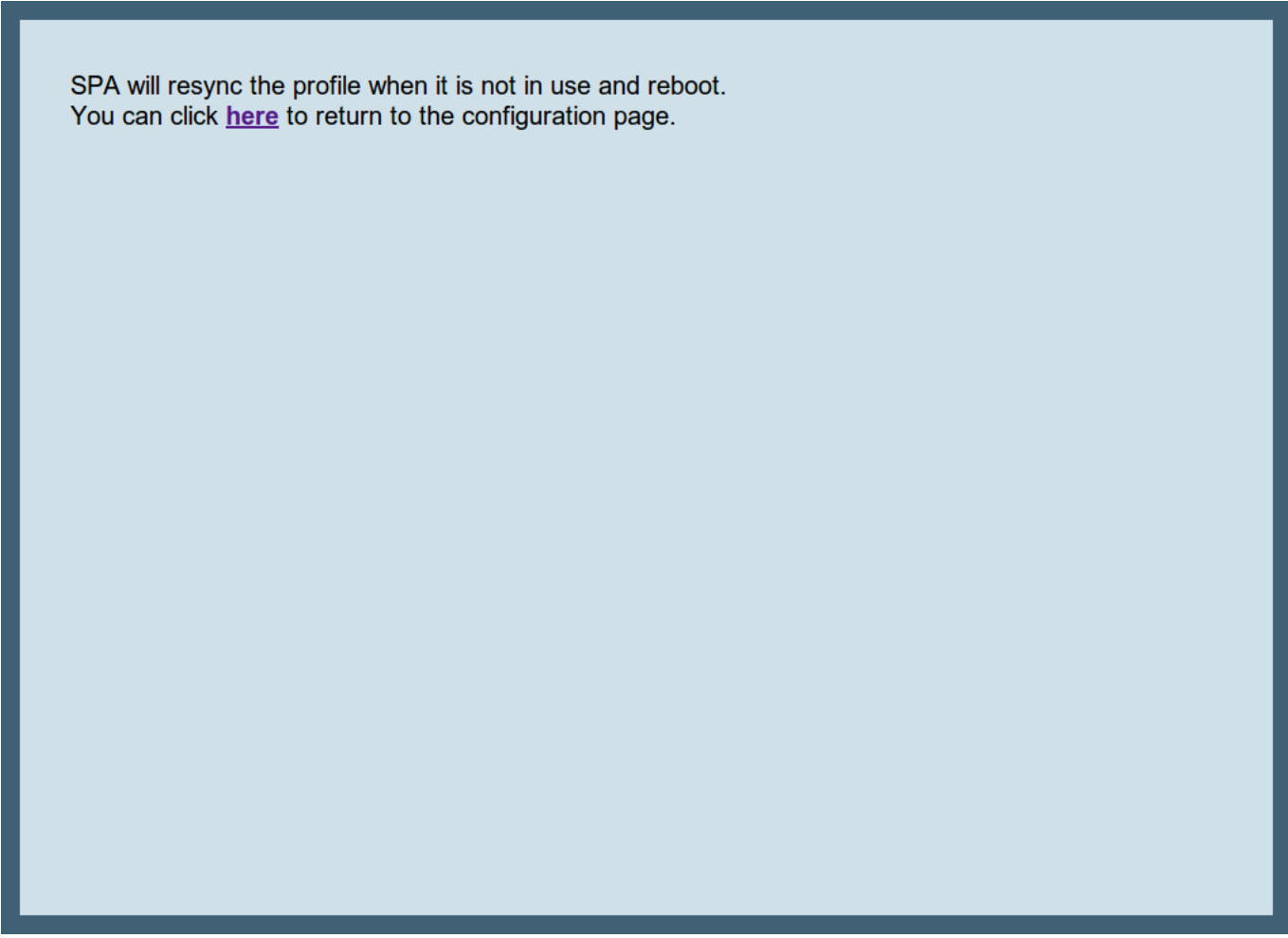


Figure 128: Sync Device

You will be redirected directly to the phone, and the Provisioning URL is automatically set. If everything goes right, you will see a confirmation page from the phone that it's going to reboot.



SPA will resync the profile when it is not in use and reboot.  
You can click [here](#) to return to the configuration page.

Figure 129: Device Sync Confirmation from Phone

You can close the browser window/tab and proceed to sync the next subscriber.

---

**Tip**

You only have to do this step once per phone to tell it the actual provisioning URL, where it can fetch the configuration from. From there, it will regularly sync with the server automatically to check for configuration changes, and applies them automatically.

---

### 16.1.6 Configuring Sound Sets for the Customer PBX

In the *Customer Details* view, there is a row *Sound Sets*, where the customer can define his own sound sets for *Auto Attendant*, *Music on Hold* and the *Office Hours Announcement*.

To create a new sound set, open the *Sound Sets* row and click *Create Sound Set*.

If you do this as administrator or reseller, the Reseller and/or Customer is pre-selected, so keep it as is. If you do this as customer, you don't see any *Reseller* or *Customer* fields.

So the important settings are:

- **Name:** The name of the sound set as it will appear in the *Subscriber Preferences*, where you can assign the sound set to a subscriber.
- **Description:** A more detailed description of the sound set.
- **Default for Subscribers:** If this setting is enabled, then the sound set is automatically assigned to all already existing subscribers which do NOT have a sound set assigned yet, and also for all newly created subscribers.

Fill in the settings and click *Save*.

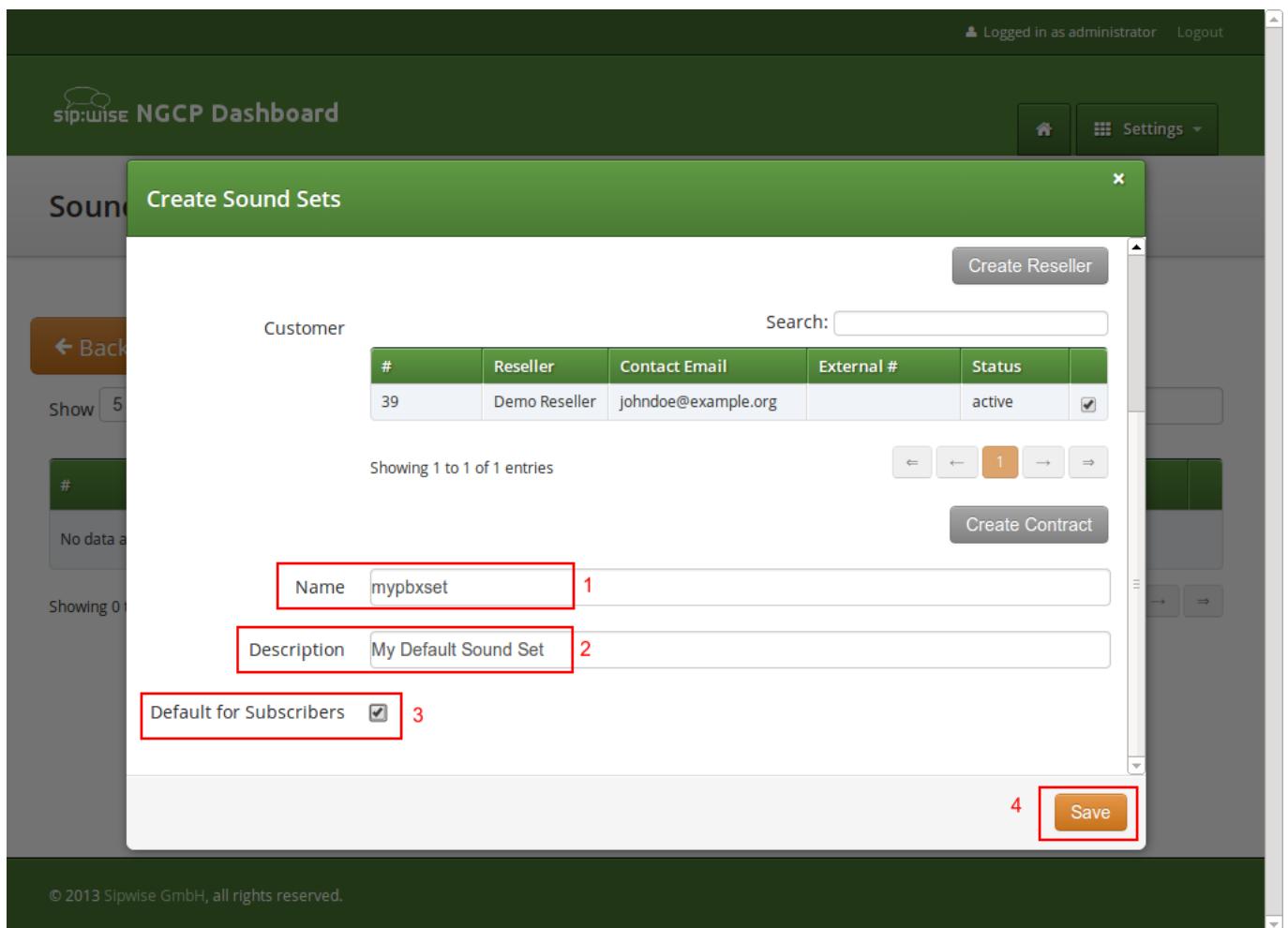


Figure 130: Create Customer Sound Set

To upload files to your Sound Set, click the *Files* button for the Sound Set.

### 16.1.6.1 Uploading a Music-on-Hold File

Open the *music\_on\_hold* row and click *Upload* on the *music\_on\_hold* entry. Choose a WAV file from your file system, and click the *Loopplay* setting if you want to play the file in a loop instead of just once. Click *Save* to upload the file.

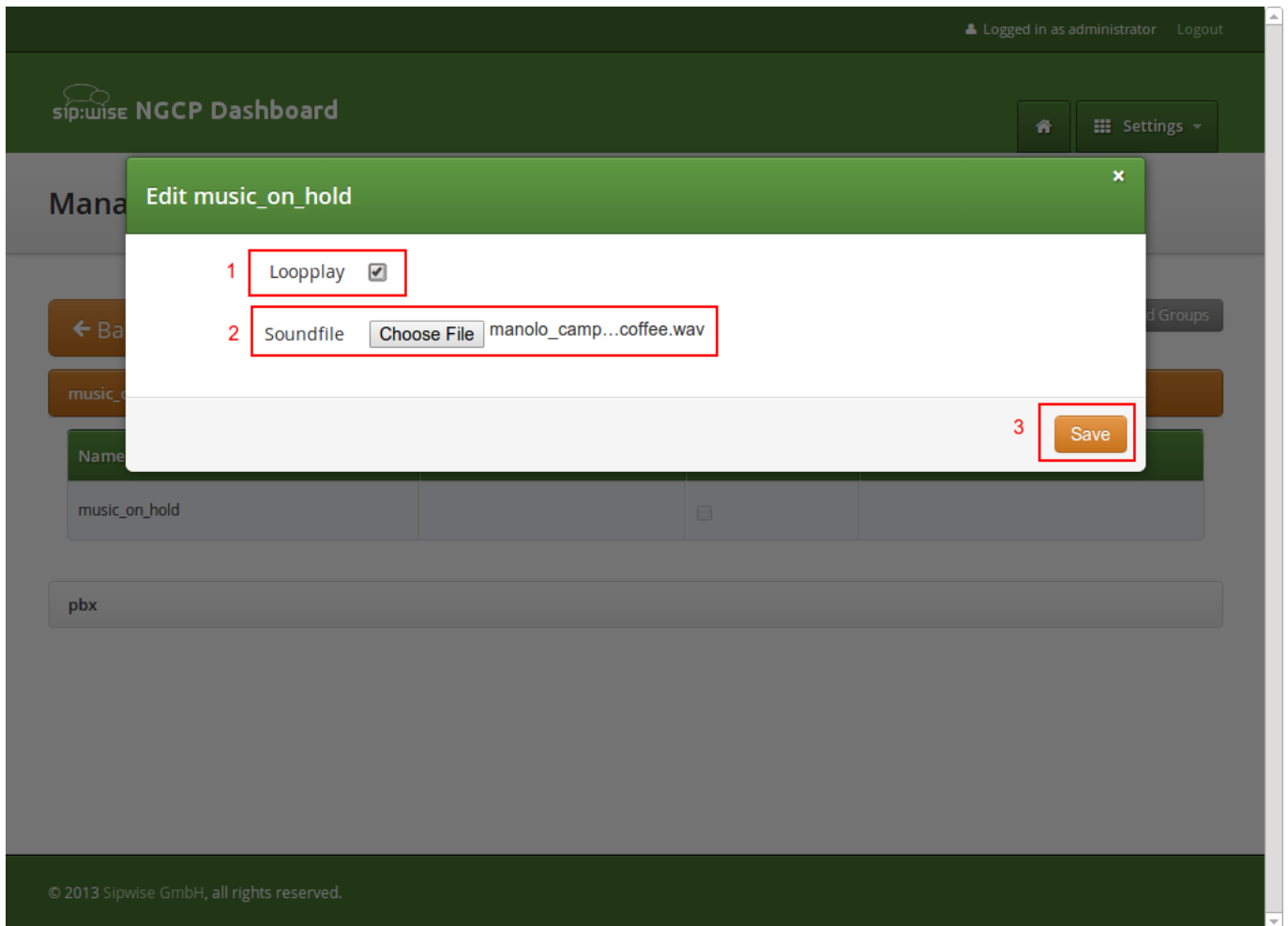


Figure 131: Upload MoH Sound File

### 16.1.7 Auto-Attendant Function

The *Auto-Attendant* is a built-in IVR feature that is available to Cloud PBX subscribers. It provides an automatic voice menu that enables the caller to select from a number of destinations, which could be other PBX subscribers or groups.

Another typical use case for the *Auto-Attendant* function is when the customer would like to have an "office assistant" that automatically takes incoming calls and routes them to the desired extension (i.e. to a subscriber).

The *Auto-Attendant* offers 2 ways of selecting the final call destination:

- *option selection*: selecting one of the pre-configured destinations by pressing a single digit (0-9)
- *extension dialing*: entering an arbitrary PBX extension number directly

#### 16.1.7.1 Enabling the Auto-Attendant

The Auto-Attendant feature can be activated for any subscriber in the Customer PBX individually. There are three steps involved:

1. You have to prepare a *Sound Set* to have Auto-Attendant sound files.
2. You have to configure the destinations for the various options you provide (e.g. pressing 1 should go to the `marketing` subscriber, 2 to `development` and 3 to some external number).
3. You have to set a Call Forward to the Auto-Attendant.

To do so, go to *Customer Details* and in the *Subscribers* section, click the *Preferences* button of the subscriber, where the Auto-Attendant should be set.

### 16.1.7.2 Preparing the Sound Set

Create a Sound Set and upload the Sound Files for it as described below. Afterwards in the *Subscriber Preferences* view, set the *Customer Sound Set* preference to the Sound Set to be used. To do so, click *Edit* on the *Customer Sound Set* preference and assign the set to be used.

#### **Uploading Auto-Attendant Sound Files**

When configuring a Call Forward to the *Auto-Attendant*, it will play the following files:

- `aa_welcome`: This is the welcome message (the greeting) which is played when someone calls the Auto-Attendant.
- each available pair of `aa_X_for/aa_X_option`: Each menu item in the Auto-Attendant consists of two parts. The `for` part, which plays something like *Press One for*, and the `option` part, which play something like *Marketing*. The Auto-Attendant only plays those menu options where both the `for` part and the `option` part is present, so if you only have 3 destinations you'd like to offer, and you want them to be on keys 1, 2 and 3, you have to upload files for `aa_1_for`, `aa_1_option`, `aa_2_for`, `aa_2_option` and `aa_3_for` and `aa_3_option`.



#### **Important**

The sound files only define the general structure of what is being played to the caller. The actual destinations behind your options are configured separately in [Configuring the Auto-Attendant Slots](#) Section 16.1.7.4.

---

An example configuration could look like this:

← Back
Expand Groups

Sound handle successfully uploaded

music\_on\_hold

pbx

Name	Filename	Loop	
aa_welcome	welcome.wav	<input type="checkbox"/>	
aa_1_for	press-1.wav	<input type="checkbox"/>	
aa_1_option	for-sales.wav	<input type="checkbox"/>	
aa_2_for	press-2.wav	<input type="checkbox"/>	
aa_2_option	for-service.wav	<input type="checkbox"/>	
aa_3_for	press-3.wav	<input type="checkbox"/>	
aa_3_option	for-tech-support.wav	<input type="checkbox"/>	
aa_4_for		<input type="checkbox"/>	
aa_4_option		<input type="checkbox"/>	
aa_5_for		<input type="checkbox"/>	

Figure 132: Upload Auto-Attendant Options Sound Files

In order to activate the **extension dialing** function within the Auto-Attendant, you have to upload the following prompt files:

- aa\_star\_for, aa\_star\_option: the announcement "Press star for connecting to an extension" (or similar message, depending on customer's needs)
- aa\_enter\_extension: will instruct the caller to enter the phone number of the extension he wants to connect to
- aa\_invalid\_extension: will be played when the phone number entered does not match any of the customer's extensions



aa_8_option		<input type="checkbox"/>
aa_9_for		<input type="checkbox"/>
aa_9_option		<input type="checkbox"/>
aa_enter_extension		<input type="checkbox"/>
aa_invalid_extension		<input type="checkbox"/>
aa_star_for		<input type="checkbox"/>
aa_star_option		<input type="checkbox"/>
aa_welcome		<input type="checkbox"/>
office_hours		<input type="checkbox"/>
queue_full		<input type="checkbox"/>

Figure 133: Upload Auto-Attendant Extension Dialing Sound Files

### 16.1.7.3 Auto-Attendant Flowchart with Voice Prompts

The illustration below shows the sequence of voice prompts played when Auto-Attendant feature is activated and a caller listens the IVR menu.

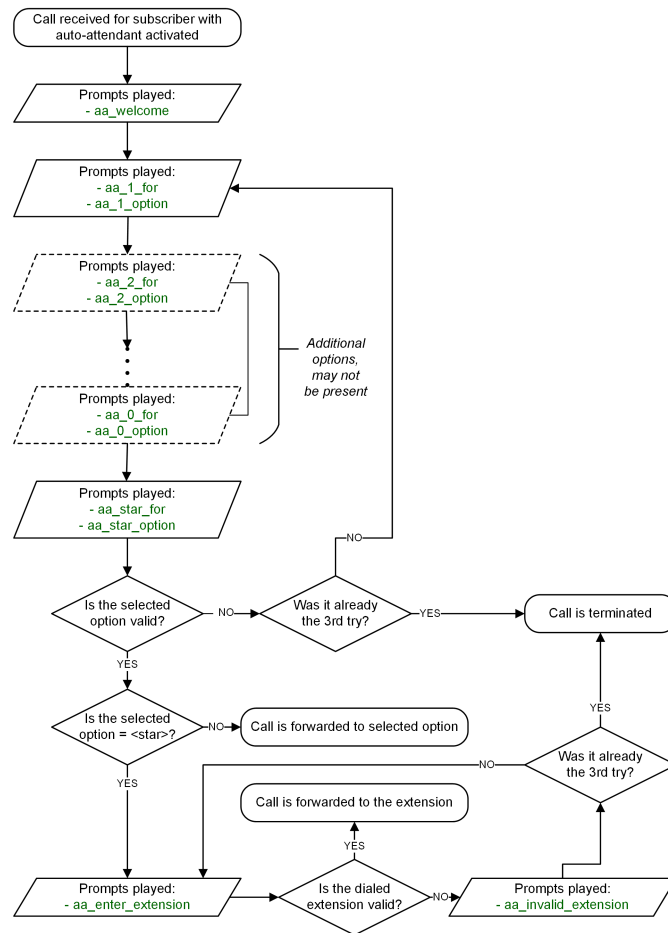


Figure 134: Flowchart of Auto-Attendant

#### 16.1.7.4 Configuring the Auto-Attendant Slots

In the *Auto-Attendant Slots* section, click the *Edit Slots* button to configure the destination options. There are up to 10 available slots to configure, from keys 0 to 9.

**Tip**

Be aware that only configured slots will be prompted in the Auto-Attendant menu.

Click *Add another Slot* to add a destination option, select the Key the destination should be assigned to, and enter a Destination. The destination can be a subscriber username (e.g. marketing), a full SIP URI (e.g. sip:michelle.miller@custom.eri.pbx.example.org or any external SIP URI) or a number or extension (e.g. 491234567 or 101).

Repeat the step for every option you want to add, then press *Save*.

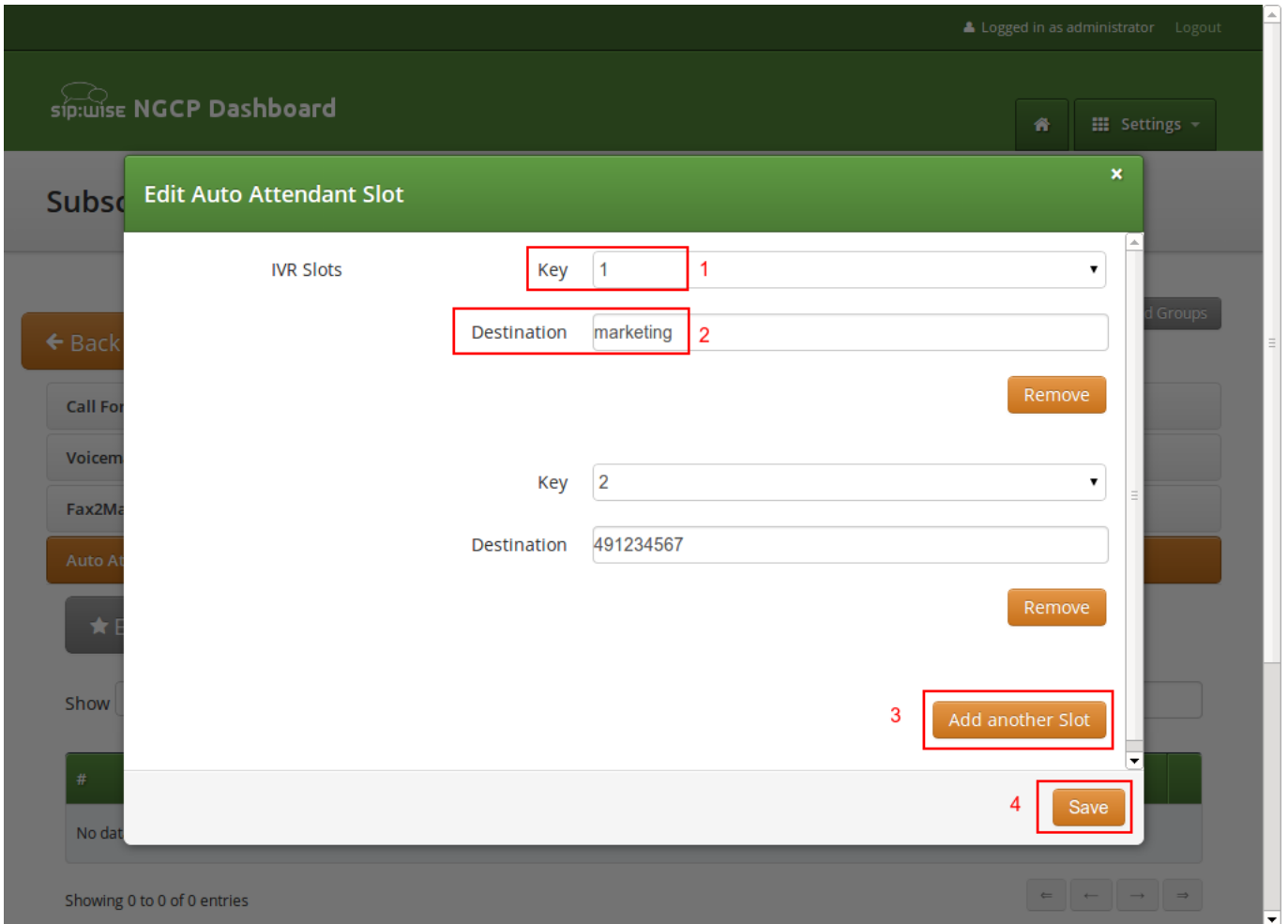


Figure 135: Define the Auto-Attendant Slots

### 16.1.7.5 Activating the Auto-Attendant

Once the Sound Set and the Slots are configured, activate the Auto-Attendant by setting a Call Forward to Auto-Attendant.

To do so, open the *Call Forwards* section in the *Subscriber Preferences* view and press *Edit* on the Call Forward type (e.g. *Call Forward Unconditional* if you want to redirect callers unconditionally to the Auto-Attendant).

Select *Auto-Attendant* and click *Save* to activate the Auto-Attendant.

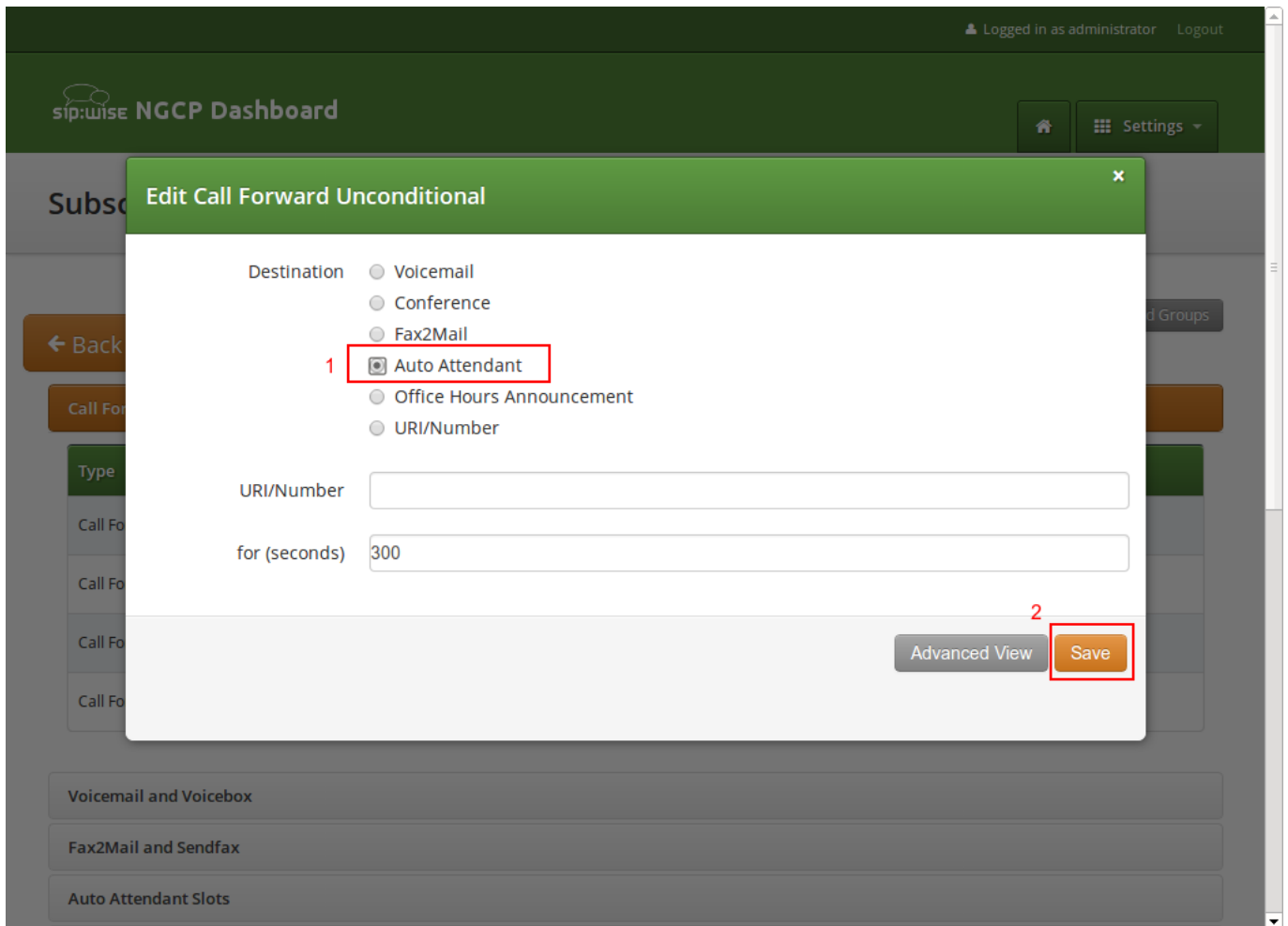


Figure 136: Set a Call Forward to Auto-Attendant

**Tip**

As with any other Call Forward, you can define more complex forwarding rules in the *Advanced View* to only forward the call to the Auto-Attendant during specific time periods, or as a fallback if no one picks up the office number.

**16.1.8 Cloud PBX Groups with Busy Members**

A *huntgroup* or a *PBX Group* is a Cloud PBX feature that distributes the calls between members of the group according to the configured hunt policy and timeout. The PBX group belongs to a customer and one Cloud PBX subscriber can be a member of one or more of the huntgroups of the customer. *Call Waiting* is a CPE (phone) feature that allows you to take another call while you're already on the phone.

Multiple incoming calls to the huntgroup may result in multiple calls delivered to the same subscriber if the Call Waiting feature is enabled on his phone, regardless whether the huntgroup members are busy at this time. Hence, busy subscribers may get a second incoming call. It may be an expected behavior (since one subscriber may have multiple devices and/or clients that all ring in parallel) or not, depending on the setup.

Therefore, the sip:carrier Cloud PBX module offers *Skip busy huntgroup members* feature to check the busy status of individual huntgroup members before routing a call to them. This will leave subscribers on active phone calls undisturbed by calls to huntgroup.

The configuration of the *Skip busy huntgroup members* feature is done via the main configuration file: `/etc/ngcp-config/config.yml`. The relevant section is: `kamailio.proxy.pbx.skip_busy_hg_members`, the example below shows the default values of the parameters.

```
skip_busy_hg_members:
  enable: 'no'
  redis_key_name: 'totaluser'
```

Option `kamailio.proxy.pbx.skip_busy_hg_members.enable` determines if call destined to a huntgroup is routed to subscribers that have busy status. When enabled and huntgroup member is busy according to the active calls information in internal Redis storage the huntgroup call is not offered to this huntgroup member. The sip:carrier platform tries the other available HG members.

**Important**

This option does not present an extended server-side Call Waiting functionality. It concerns only the huntgroups' behavior. Hence subscriber would still be able to receive multiple calls when called directly (not via huntgroup) with Call Waiting enabled on his phone.

---

The option `redis_key_name` may take the following values:

- *totaluser*: The callee is busy when involved in one or more incoming or outgoing calls in active or alerting phase.
- *activeuser*: The callee is busy when involved in one or more incoming or outgoing calls in active or alerting phase but NOT busy for the calls that are forwarded.

When the feature is enabled with `redis_key_name` set to *totaluser*:

```
skip_busy_hg_members:
  enable: 'yes'
  redis_key_name: 'totaluser'
```

The behavior when calling the huntgroup is the following:

- The callee is busy when involved in one or more incoming or outgoing calls in active or alerting phase.
- The callee is busy for incoming calls that are forwarded.

This can be better illustrated by the following use cases:

**Use Case 1**

Subscriber receives an incoming call. A second call is made to the HG. The subscriber should NOT receive this call via HG extension.

### Use Case 2

Subscriber makes an outgoing call. A second call is made to the HG. The subscriber should NOT receive this call via HG extension.

### Use Case 3

Subscriber with call forwards (CFU, CFB, CFNA, CFT) receives a call to his extension (not extension of HG) which is then forwarded. A second call is made to the HG. The subscriber should NOT receive the call via HG extension.

In order to prevent the forwarded calls from keeping the subscriber as "busy" for the purpose of this feature the platform administrator should set the `kamailio.proxy.pbx.skip_busy_hg_members.redis_key_name` parameter to value `activeuser`:

```
skip_busy_hg_members:
  enable: 'yes'
  redis_key_name: 'activeuser'
```

While User Cases 1 and 2 will behave in the same way as described above, the change of behavior happens in Use Case 3:

### Use Case 3

Subscriber with call forwards (CFU, CFB, CFNA, CFT) receives a call to his extension (not extension of HG) which is then forwarded. A second call is made to the HG. The subscriber should receive the call as normal.

There is a possibility to fine-tune when callee is considered busy and exclude, for example, intra-PBX calls or calls to voicemail from keeping subscriber as "busy". Please contact Sipwise support if you'd like to do that.

## 16.1.9 Configuring Call Queues

The sip:carrier platform offers call queueing feature for Cloud PBX subscribers. For any subscriber within the PBX the NGCP system administrator or the subscriber himself may activate the *Call Queue*. This is done individually for each subscriber on demand.

If call queue activation has been done and the subscriber receives more than 1 call at a time, then the second and all further callers will be queued until the subscriber finishes his call with the first caller and gets free.

### 16.1.9.1 Activating the Call Queue

The call queue configuration is available at the path: *Subscribers* → *select one* → *Details* → *Preferences* → *Cloud PBX*.

Following configuration parameters may be set for call queueing:

- `cloud_pbx_callqueue` : shows the status of call queueing (enabled / disabled); by default it is disabled
- `max_queue_length` : the length of call queue, i.e. the maximum number of callers in a queue; the default is 5
- `queue_wrap_up_time` : the delay in seconds between the ending of the previous call and the connection of the next queued caller with the subscriber; the default is 10

In order to change the actual setting, press the *Edit* button in the relevant row.

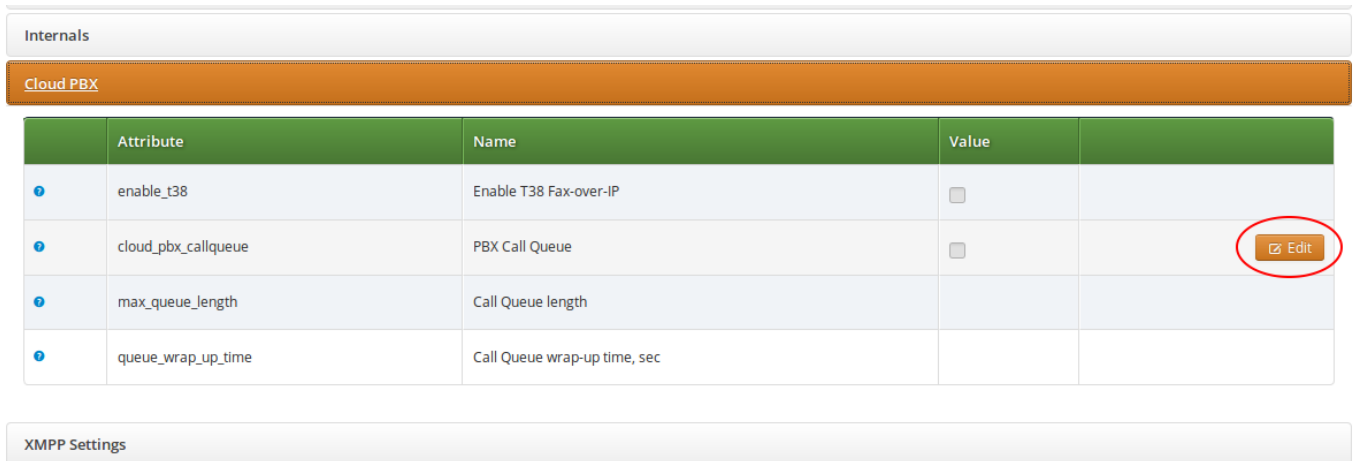


Figure 137: Call Queue Configuration

### 16.1.9.2 Call Queue Voice Prompts

Queued callers first hear a greeting message then information about their position in the queue and finally a waiting music / signal.

Table 23: Call Queue Voice Prompts

Prompt handle	Prompt content
queue_greeting	All lines are busy at the moment, you are being queued.
queue_prefix	You are currently number. . .
queue_suffix	. . . in the queue, please hold the line.
queue_full	All lines are busy at the moment, please try again later.
queue_waiting_music	<waiting music>

### 16.1.9.3 Call Queue Flowchart with Voice Prompts

The following illustration shows which voice prompts are played to the caller when the call gets into a queue.

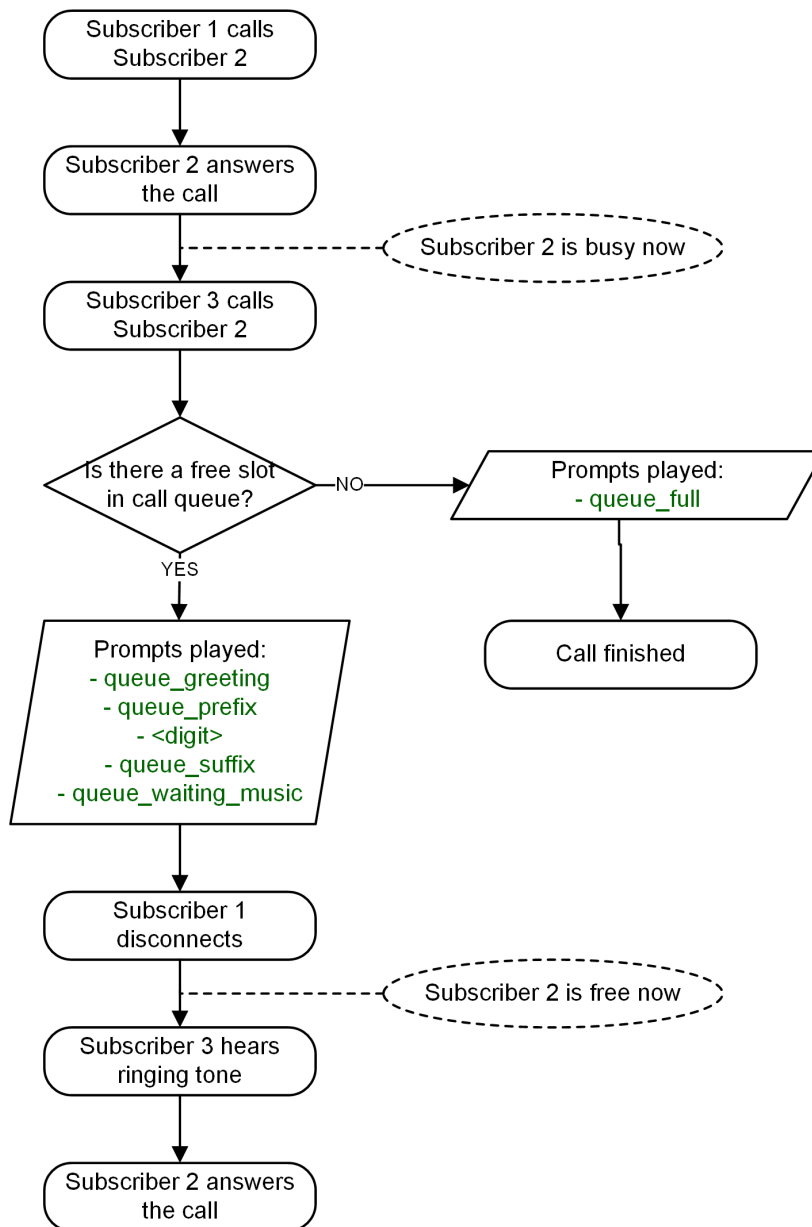


Figure 138: Flowchart of Call Queue

### 16.1.10 Device Auto-Provisioning Security

#### 16.1.10.1 Server Certificate Authentication

The Cisco SPA phones can connect to the provisioning interface of the PBX via HTTP and HTTPS. When perform secure provisioning over HTTPS, the phones validate the server certificate to check if its a legitimate Cisco provisioning server. To pass this check, the provisioning interface must provide a certificate signed by Cisco for that exact purpose.

The following steps describe how to obtain such a certificate.

First, a new SSL key needs to be generated:



```
$ openssl genrsa -out provisioning.key 2048
Generating RSA private key, 2048 bit long modulus
...+++
.....+++
e is 65537 (0x10001)
```

Next, a certificate signing request needs to be generated as follows. Provide your company details.



#### Important

The **Common Name (e.g. server FQDN or YOUR name)** field is crucial here. Provide an FQDN which the phones will later use via DNS to connect to the provisioning interface, for example *pbx.example.org*. Cisco does **NOT** support wild-card certificates.



#### Important

Leave the password empty when asked for it (press Enter without entering anything).

```
$ openssl req -new -key provisioning.key -out provisioning.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
```

```
Country Name (2 letter code) [AU]:AT
State or Province Name (full name) [Some-State]:Vienna
Locality Name (eg, city) []:Vienna
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Sipwise GmbH
Organizational Unit Name (eg, section) []:Operations
Common Name (e.g. server FQDN or YOUR name) []:pbx.example.org
Email Address []:office@sipwise.com
```

```
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Finally, compress the `provisioning.csr` file via ZIP and send it to our Cisco sales representative. If in doubt, you can try to send it directly to `ciscosb-certadmin@cisco.com` asking them to sign it.



#### Important

Only send the CSR file. **Do NOT send the key file, as this is your private key!**

**Important**

Ask for both the signed certificate AND a so-called *combinedca.crt* which is needed to perform client authentication via SSL. Otherwise you can not restrict access to Cisco SPAs only.

---

You will receive a signed CRT file, which Sipwise can use to configure the PBX provisioning interface.

### 16.1.10.2 Client Certificate Authentication

If a client connects via HTTPS, the server also checks for the client certificate in order to validate that the device requesting the configuration is indeed a legitimate Cisco phone, and not a fraudulent user with a browser trying to fetch user credentials.

Cisco Client Root Certificate can be obtained from [Download Client Certificates](#) website.

### 16.1.11 Device Bootstrap and Resync Workflows

The IP phones supported by the PBX need to initially be configured to fetch their configuration from the system. Since the phones have no initial information about the system and its provisioning URL, they need to be boot-strapped. Furthermore, changes for a specific device might have to be pushed to the device immediately instead of waiting for it to re-fetch the configuration automatically.

The following sections describe the work-flows how this is accomplished without having the customer directly accessing the phone.

#### 16.1.11.1 Cisco SPA Device Bootstrap

##### *Initial Bootstrapping*

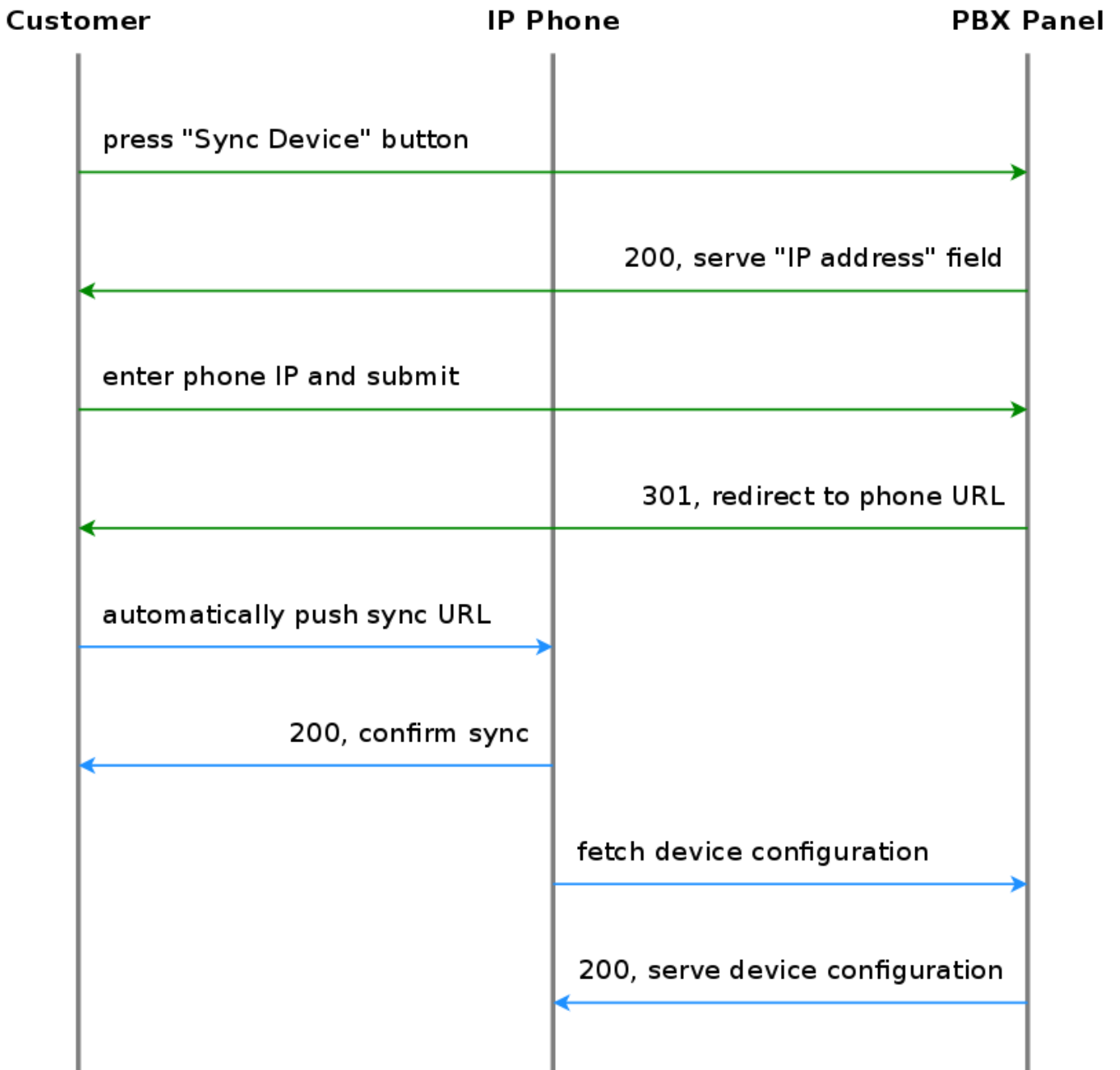


Figure 139: Initially bootstrap a PBX device

**Subsequent Device Resyncs**

If one of the subscribers configured on a PBX device is registered via SIP, the system can trigger a re-sync of the phone directly via SIP without having the customer enter the IP of the phone again. This is accomplished by sending a special NOTIFY message to the subscriber:

```

NOTIFY sip:subscriber@domain SIP/2.0
To: <sip:subscriber@domain>
From: <sip:subscriber@domain>;tag=some-random-tag
  
```

```

Call-ID: some-random-call-id
CSeq: 1 NOTIFY
Subscription-State: active
Event: check-sync
Content-Length: 0
    
```

In order to prevent unauthorized re-syncs, the IP phone challenges the request with its own SIP credentials, so the NOTIFY is sent twice, once without authentication, and the second time with the subscriber's own SIP credentials.

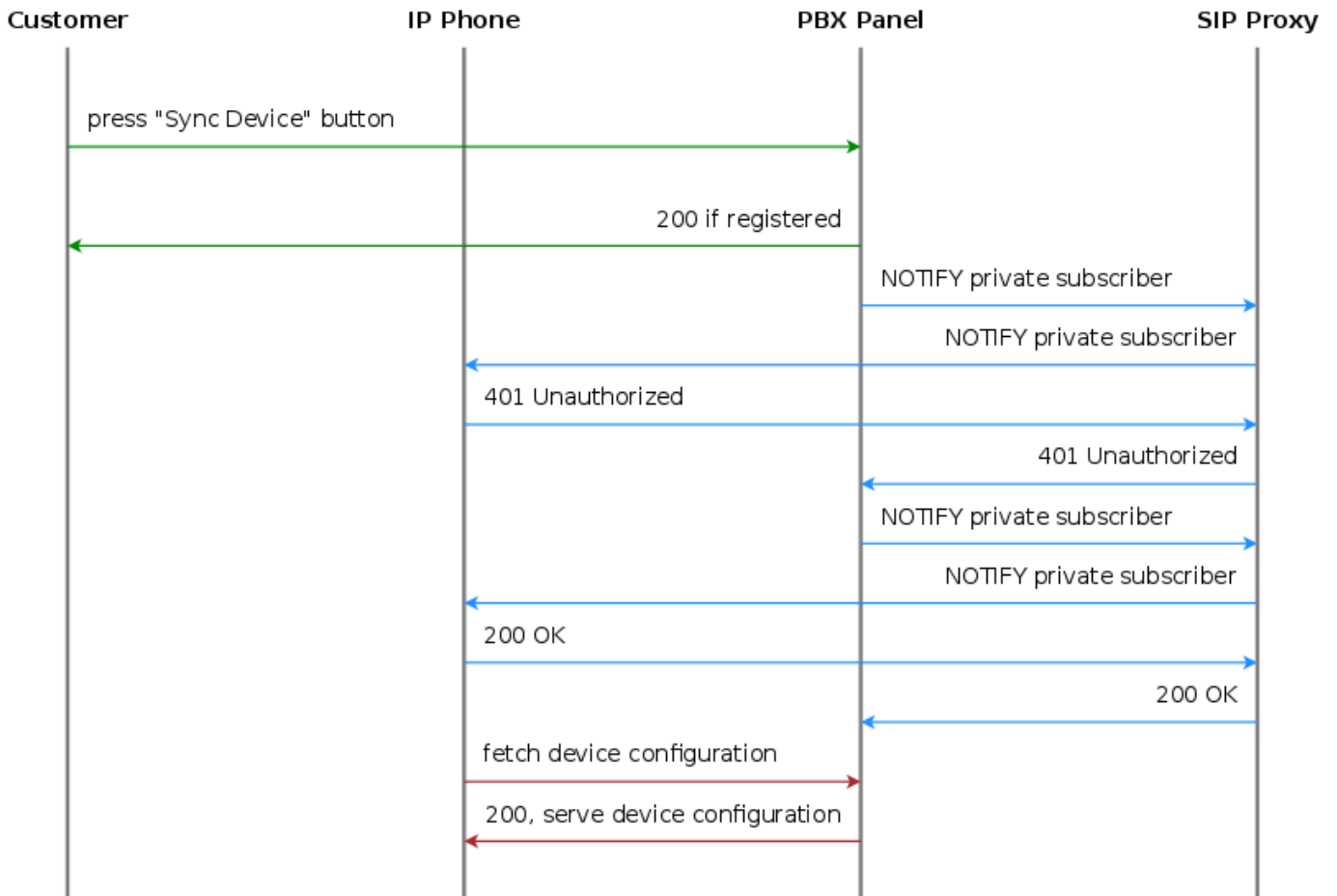


Figure 140: Resync a registered PBX device

### 16.1.11.2 Panasonic Device Bootstrap

#### Initial Bootstrapping

Panasonic provides a zero-touch provisioning mechanism in their firmwares, which causes the factory-reset phones to connect to a Panasonic web service at <https://provisioning.e-connecting.net> to check if a custom provisioning URL is configured for the MAC address of the phone. If an association between the MAC and a provisioning URL is found, the web service redirects the phone to the provisioning URL, where the phone connects to in order to obtain the configuration file.

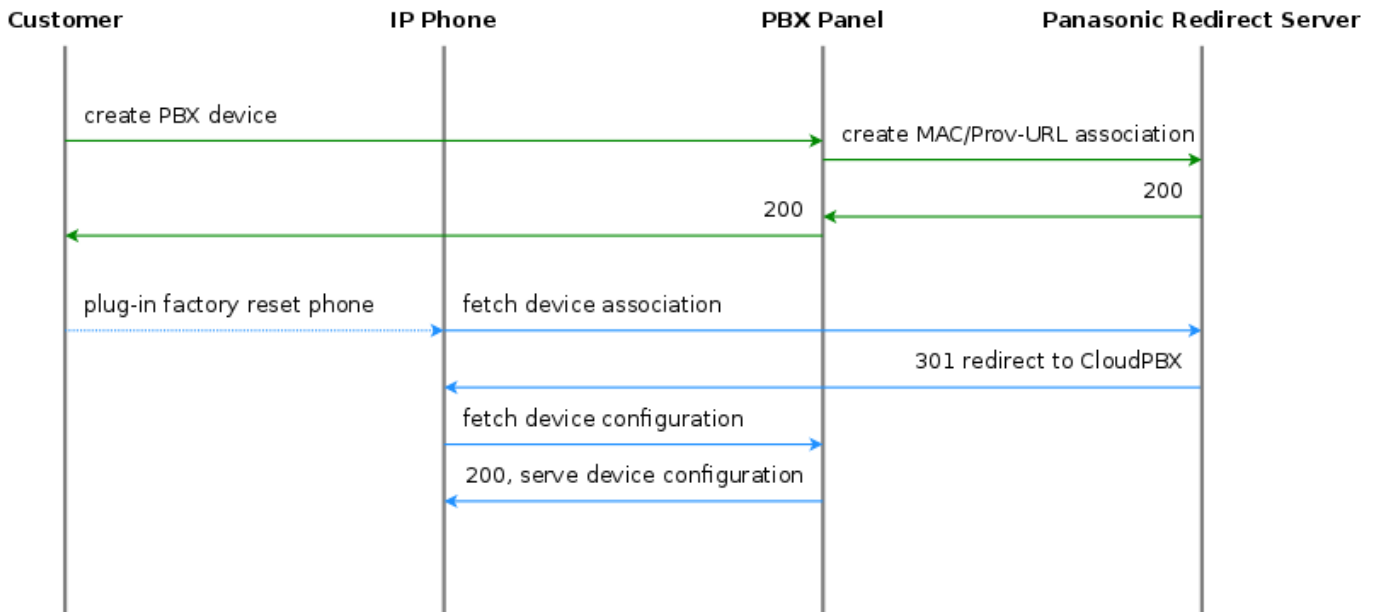


Figure 141: Initially bootstrap a Panasonic phone

The CloudPBX module ensures that when an end customer creates a Panasonic device, the MAC address is automatically provisioned on the Panasonic web service via an API call, so the customer's phone can use the correct provisioning URL to connect to the auto-provisioning server of the CloudPBX.

As a result, no customer interaction is required to bootstrap Panasonic phones, other than just creating the phone with the proper MAC on the CloudPBX web interface.

### **Factory Reset**

For already provisioned phones, the end customer might need to perform a factory reset:

- Press *Settings* or *Setup*
- Enter *#136*
- Select *Factory Setting* and press *Enter*
- Select *Yes* and press *Enter*
- Select *Yes* and press *Enter*

The default username for factory-reset phones is *admin* with password *adminpass*.

### **Subsequent Device Resyncs**

The same procedure as with Cisco SPA phones applies, once a subscriber configured on the phone is registered.

### **16.1.11.3 Yealink Device Bootstrap**

#### **Initial Bootstrapping**

Yealink provides a zero-touch provisioning mechanism in their firmwares, which causes the factory-reset phones to connect to a Yealink web service at <https://rps.yealink.com> to check if a custom provisioning URL is configured for the MAC address of the phone. If an association between the MAC and a provisioning URL is found, the web service redirects the phone to the provisioning URL, where the phone connects to in order to obtain the configuration file.

If both Cisco SPA and Yealink phones are used, an issue with the Cisco-signed server certificate configured on the provisioning port (1444 by default) of the CloudPBX provisioning server arises. Yealink phones by default only connect to trusted server certificates, and the Cisco CA certificate used to sign the server certificate is not trusted by Yealink. Therefore, a two-step approach is used to disable the trusted check via a plain insecure http port (1445 by default) first, where only device-generic config options are served. No user credentials are provided in this case, because no SSL client authentication can be performed. The generic configuration disables the trusted check, and at the same time changes the provisioning URL to the secure port, where the Yealink phone is now able to connect to.

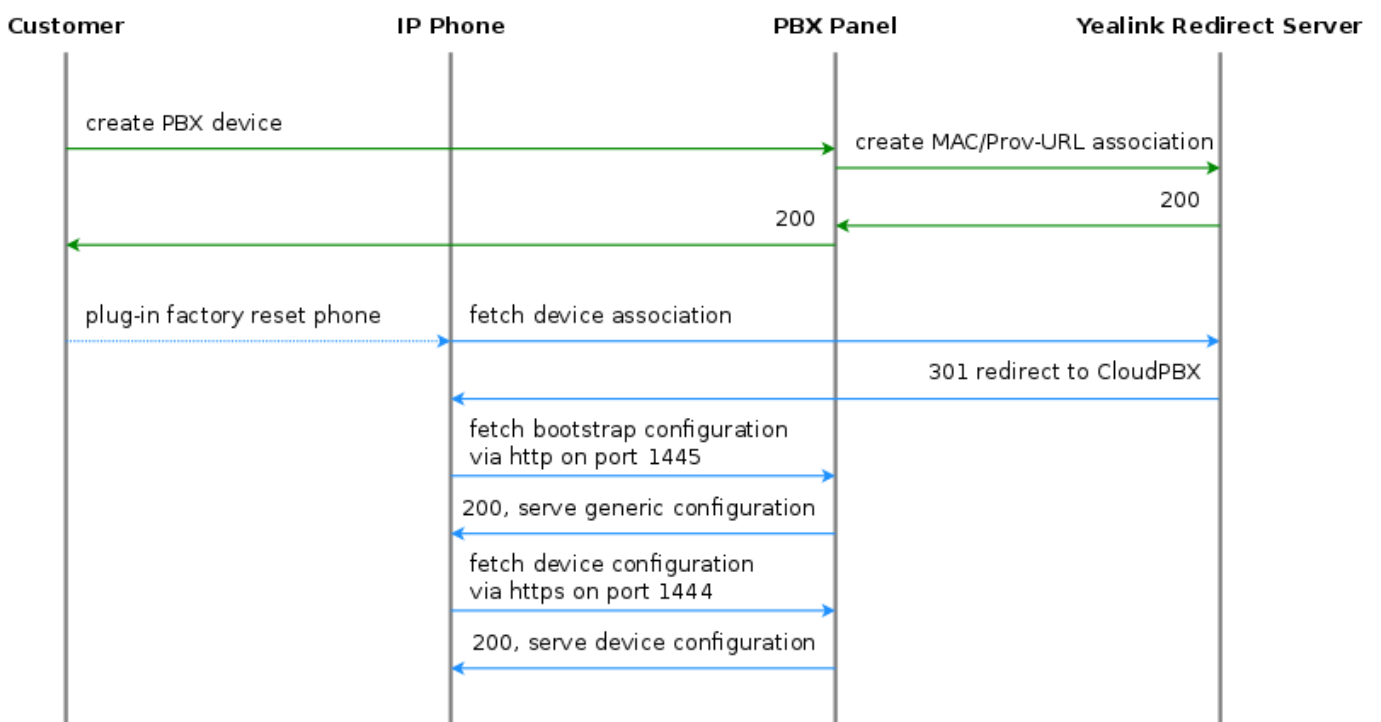


Figure 142: Initially bootstrap a Yealink phone

The CloudPBX module ensures that when an end customer creates a Yealink device, the MAC address is automatically provisioned on the Yealink web service via an API call, so the customer’s phone can use the correct insecure bootstrap provisioning URL to connect to the auto-provisioning server of the CloudPBX for the generic configuration, which in turn provides the information on where to connect to for the secure, full configuration.

As a result, no customer interaction is required to bootstrap Yealink phones, other than just creating the phone with the proper MAC on the CloudPBX web interface.

**Factory Enable Yealink Auto-Provisioning**

Older Yealink firmwares don’t automatically connect to the Yealink auto-provisioning server on initial boot, so it needs to be enabled manually by the end customer.

- Log in to `http://phone-ip/servlet?p=hidden&q=load` using `admin` and `admin` as user/password when prompted
- Change `Redirect Active` to `Enabled`
- Press `Confirm` and power-cycle phone

### **Subsequent Device Resyncs**

The same procedure as with Cisco SPA phones applies, once a subscriber configured on the phone is registered.

#### **16.1.11.4 Audiocodes Mediant Device Bootstrap and Configuration**

##### **Initial Bootstrapping**

An Audiocodes device provides a zero-touch provisioning mechanism in its firmware which causes a factory-reset device to connect to the URL built into the firmware. This URL is pointing to the NGCP provisioning server (in case of NGCP Carrier: `web01` node) listening on TCP port 1444 for HTTPS sessions.

The prerequisites for the device provisioning are that the device has a routable IP address and can reach the IP address of the NGCP provisioning interface.

The Audiocodes device should request the firmware file or CLI configuration file from the NGCP platform. The firmware versions and CLI config versions are decoupled from each other; the NGCP can not enforce specific version of the firmware on the device. Instead, it should be requested by the device itself. In other words, provisioning is a *pull* and not a *push* process.

NGCP expects the provisioning request from the Audiocodes device after SSL handshake and serves the requested file to the device if the device provides valid MAC address as the part of the URL. The MAC address is used to identify the device to the NGCP platform. The firmware and CLI config files are provided at the following URLs:

- the base URL to download firmwares: `https://<NGCP_IP>:1444/device/autoprov/firmware/001122334455/455/from/0/latest`
- the base URL to download CLI config: `https://<NGCP_IP>:1444/device/autoprov/config/001122334455`

where 001122334455 should be replaced with the actual device's MAC address and `<NGCP_IP>` with IP address of the NGCP provisioning interface.

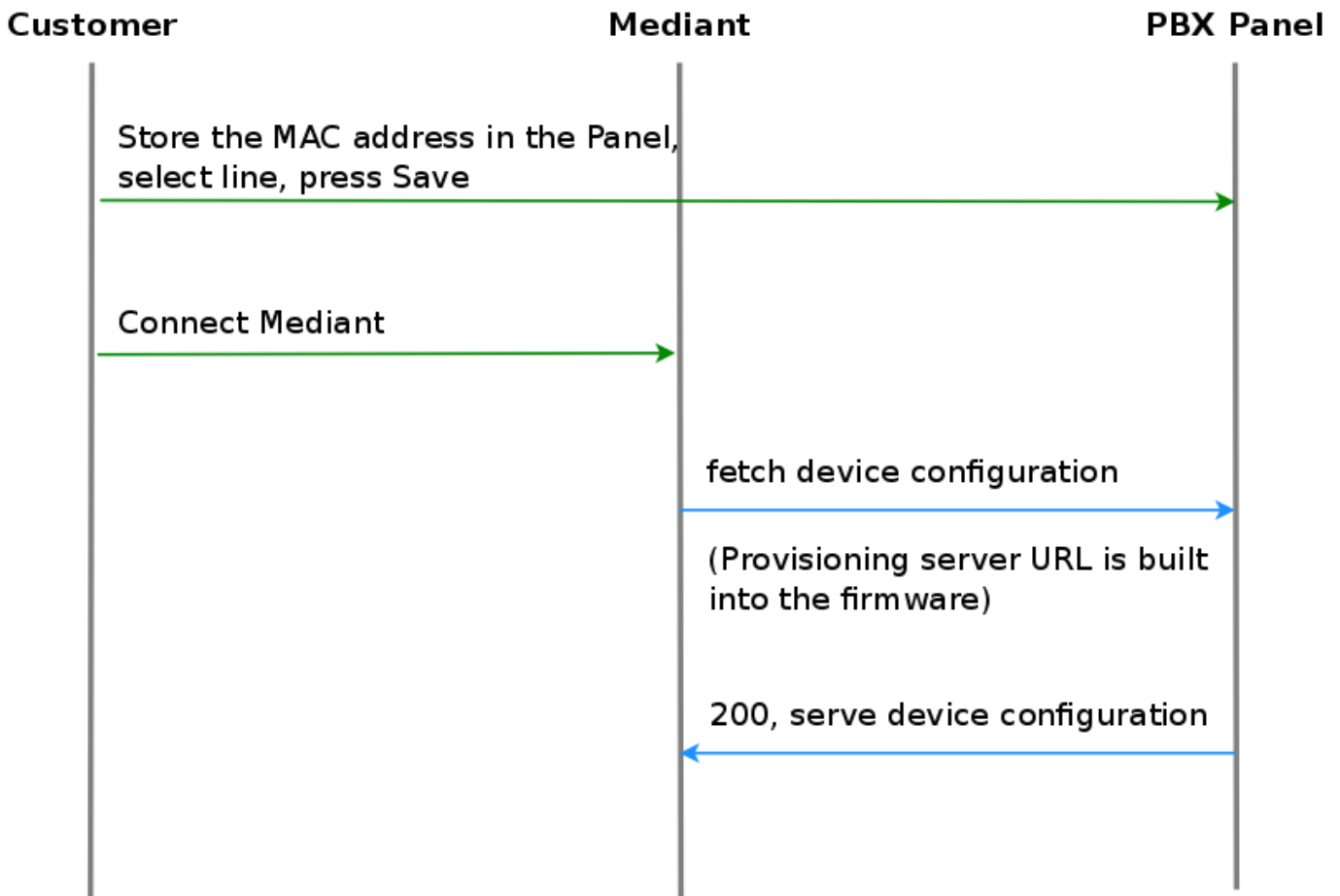


Figure 143: Initially bootstrap a Mediant gateway

**Device management basics**

The list of device models, firmwares and configurations are global to a reseller and are available for end customer. This data is initially provided by Sipwise as bulk upload of all supported phone models. The firmwares and settings are stored in the database on the DB node pair(s). The NGCP leverages the Cloud PBX module with its template system to generate the configurations and firmware files from database on the fly. Please refer to the following chapters in NGCP handbook for the current information on how to perform device management:

- [Uploading device firmwares](#) Section 16.1.1.2
- [Creating device configuration](#) Section 16.1.1.3
- [Creating device profiles](#) Section 16.1.1.4

**Parameterizing the Device Configuration Template**

The device-specific parameters are filled in by the system individually when a physical device fetches its configuration file. Parameters from the NGCP panel:

- `username`: Subscriber Details → Master Data → SIP Username



- `password`: Subscriber Details → Master Data → SIP Password
- `domain`: Subscriber Details → Master Data → Domain
- `extension`: Subscriber Details → Master Data → Extension
- `area code`: Subscriber Preferences → Number Manipulations → ac
- `country code`: Subscriber Preferences → Number Manipulations → cc

The produced **CLI config file** has the following structure:

1. SIP account credentials:

```
"sip-definition account 0"
```

- `user-name` [username]
- `password` [password]
- `host-name` [domain]
- `register` reg
- `contact-user` "[country code][area code][extension]"

2. IP Groups:

```
"voip-network ip-group 1" and "voip-network ip-group 2"
```

- `sip-group-name` [domain]

3. Proxy and registration settings:

```
"sip-definition proxy-and-registration"
```

- `set gw-name` [domain]

4. Manipulations:

- `manipulation-name` "from trunk domain":

```
"sbc manipulations message-manipulations 3"
```

```
- action-value "[% line.domain %]"
```

- `manipulation-name` "clip no screening":

```
"sbc manipulations message-manipulations 8"
```

```
- action-value "'<sip:+[country code][area code][extension]@' + param.ipg.dst.host + '
>' "
```

*Specific CLI parameters are:*

- [IPPBX\_Hostname]
- [IPPBX\_server\_IP]

which are used at the following configuration parameters:

- Proxy settings:

```
"voip-network proxy-ip 1"
```

```
– proxy-address [IPPBX_Hostname]
```

- Manipulations:

```
"sbc manipulations message-manipulations 1"
```

```
– action-value [IPPBX_Hostname]
```

### 16.1.12 Device Provisioning and Deployment Workflows

This chapter provides information and hints for preparing and performing the deployment of certain VoIP devices at customer sites, that have a customer-facing interface which also needs customisation.

#### 16.1.12.1 Audiocodes Mediant Device Provisioning Workflow

Audiocodes ISDN gateways and eSBCs are devices used to connect legacy (ISDN) PBX and IP-PBX to the Sipwise NGCP platform and maintain their operations within the Operator's network. Sipwise NGCP offers a *SipConnect 1.1* compliant signaling and media interface to connect SIP trunks to the platform. In addition to this interface, the Sipwise NGCP provides an auto-provisioning mechanism to configure SIP endpoints like IP phones, media gateways and eSBCs.

##### **Provisioning URL**

An Audiocodes device needs to obtain the provisioning URL of the Sipwise NGCP in one way or the other to request its device configuration and subsequently download specific firmwares, obtain SIP credentials to connect to the network facing side, and configure the customer facing side for customer devices to connect either via ISDN or SIP. Typical ways of obtaining the provisioning URL for a SIP endpoint are:

- using DHCP option-66 (in a pre-staging environment or directly at the customer premise) where vendor-specific Redirect Servers are configured in the default configuration or firmware
- getting pre-configured per deployment from the SIP endpoint vendor
- getting pre-configured per deployment by a 3rd party distributor

The assumption is that Audiocodes devices are supplied with a firmware (and all required SSL certificates) being pre-configured and the provisioning URL pointing to an Operator URL the Sipwise NGCP is serving, before handing the devices over to field service engineers doing the truck rolls.

### **Field Configuration**

The Sipwise NGCP provides a SipConnect 1.1 compliant interface on the network side for the Audiocodes devices. This interface clearly defines the numbering formats of the calling and called party, the SIP header mechanisms to provide CLI restriction, the RTP codecs, etc.

On the customer facing side, however, those variables might be different from deployment to deployment:

- An IP-PBX might choose to only send its extension as calling party number, or might choose to send the full number in national format.
- It might choose to use the SIP From-header mechanisms to suppress displaying of the CLI, or use the SIP Privacy header.
- The same uncertainty exists to some extent for a legacy PBX connecting via ISDN to the Audiocodes device.

The assumption here is that a field service engineer is NOT supposed to change the Audiocodes configuration in order to make the customer interface work, as this will lead to big issues in maintaining those local changes, especially if a replacement of the device is necessary. Instead, the Audiocodes configuration must ensure that all different kinds of variants in terms of SIP headers, codecs and number formats are translated correctly to the network side and vice versa. If it turns out that there are scenarios in the field which are not handled correctly, temporary local changes might be performed to finish a truck roll, but those changes MUST be communicated to the platform operator, and the server-side configuration templates must be adapted to handle those scenarios gracefully as well.

For deployments with ISDN interfaces on the customer facing side of the Audiocodes, different *Device Profiles* with specific *Device Configurations* per *Device Model* must exist to handle certain scenarios, specifically whether the ISDN interface is operating in Point-to-Point or Point-to-Multipoint mode. Configuration options like which side is providing the clock-rate are to be defined up-front, and the PBX must be reconfigured to adhere to the configuration.

### **Network Configuration**

On the network facing side, both the ISDN and eSBC style deployments have to be designed to obtain an IP address via DHCP. The definition of the IP address ranges is up to the Operator. It may or may not be NAT-ed, but it is advised to use a private IP range directly routed in the back-bone to avoid NAT.

On the customer facing side, networking is only relevant for the eSBC deployment. In order to make the IP-PBX configuration as stream-lined as possible, a pre-defined network should be established on the customer interface of the Audiocodes device.

---

#### **Tip**

The proposal is to define a network 192.168.255.0/24 with the Audiocodes device using the IP 192.168.255.2 (leaving the 192.168.255.1 to a possible gateway). The IP-PBX could obtain its IP address via DHCP from a DHCP server running on the Audiocodes device (e.g. serving IP addresses in the range of 192.168.255.100-254), or could have it configured manually (e.g. in the range of 192.168.255.3-99). Since the Audiocodes device IP on the customer side is always fixed at 192.168.255.2, the IP-PBX for each customer can be configured the same way, pointing the SIP proxy/registrar or outbound proxy always to this IP.

---

The customer facing side is outside the Sipwise demarcation line, that's why the network configuration mentioned above only serves as proposal and any feedback is highly welcome. However, it must be clearly communicated how the customer facing

network is going to be configured, because the Sipwise NGCP needs to incorporate this configuration into the Audiocodes configuration templates.

### 16.1.12.2 Audiocodes Mediant Device Deployment Workflow

#### *Pre-Configuration on Sipwise NGCP platform*

1. Before connecting a customer to a SIP trunk, it must be clear which Audiocodes *Device Model* is going to be used (depending on if, which and how many ISDN ports are necessary) and which *Device Profile* for the *Device Model* is required (eSBC mode, ISDN P-to-P or P-to-MP mode). Based on that, the correct physical device must be picked.
2. Next, the customer has to be created on the Sipwise NGCP. This step requires the creation of the customer, and the creation of a subscriber within this customer. For the subscriber, the proper E.164 numbers or number blocks must be assigned, and the correct subscriber preferences must be set for the network interface to adhere to the SipConnect 1.1 interface. This step is automated by a script provided by Sipwise until the provisioning work-flow is fully integrated with Operator's OSS/BSS systems. *Required parameters are:*
  - an external customer id to relate the customer entity on the Sipwise NGCP with a customer identifier in Operator's IT systems
  - a billing profile name
  - a subscriber username and password, the domain the subscriber is configured for
  - the numbers or number blocks assigned to the subscriber, and the network provided number of the subscriber
  - optional information is geographic location information and IP network information to properly map emergency calls
3. Finally, the association between the MAC address of the Audiocodes device and the SIP subscriber to be used on the SIP trunk must be established. This step is also automated by a script provided by Sipwise. *Required parameters are:*
  - the subscriber id
  - the Device Profile to be used
  - and the MAC address of the Audiocodes device

#### **Installation**

Once the above requirements are fulfilled and the customer is created on the Sipwise NGCP, the Audiocodes device can be installed at the customer premise.

When the Audiocodes device boots, it requests the configuration file from the Sipwise NGCP by issuing a GET request via HTTPS.

For **authentication and authorization** purposes, the Sipwise NGCP requests an SSL client certificate from the device and will check whether it's signed by a Certificate Authority known to the Sipwise NGCP. Therefore, Audiocodes must provide the CA certificate used to sign the devices' client certificates to Sipwise to allow for this process. Also, the Sipwise NGCP will provide an SSL server certificate to the device. The device must validate this certificate in order to prevent man-in-the-middle attacks. Options here are to have:

- Sipwise provide a self-signed certificate to Audiocodes for Audiocodes or a 3rd party distribution partner to configure it as trusted CA in the pre-staging process

- the Operator provide a certificate signed by a CA which is already in the trust store of the Audiocodes devices.

Once the secured HTTPS connection is established, the Sipwise NGCP will provide a CLI style configuration file, with its content depending on the pre-configured *Device Profile* and subscriber association to the device's MAC address.

The configuration includes the firmware version of the latest available firmware configured for the *Device Model*, and a URL defining from where to obtain it. The configuration details on how the Audiocodes devices manage the scheduling of firmware updates are to be provided by Audiocodes or its partners, since this is out of scope for Sipwise. Ideally, firmware updates should only be performed if the device is idle (no calls running), and within a specific time-frame (e.g. between 1 a.m. and 5 a.m. once a certain firmware version is reached, including some random variation to prevent all devices to download a new firmware version at the same time).

### **Device Replacement**

If a customer requires the replacement of a device, e.g. due to hardware issues or due to changing the number or type of ISDN interfaces, a new association of the new device MAC, its *Device Profile* and the subscriber must be established.

In order to make the change as seamless as possible for the customer, a new device is created for the customer with the new MAC, a proper *Device Profile*, but the same subscriber as used on the old device. Once the new device boots at the customer premise, it will obtain its configuration and will register with the same subscriber as the old device (in case it's still operational). For inbound calls to the customer, this will cause parallel ringing to take place, and it's up to the customer or the field engineer when to re-configure or re-cable the PBX to connect to one or the other device.

Once the old device is decommissioned, the old MAC association can be deleted on the Sipwise NGCP.

#### **16.1.13 List of available pre-configured devices**

<b>Vendor</b>	<b>Model</b>	<b>Available from release</b>
Audiocodes	Mediant800	mr4.1.1.1
Cisco	ATA112	mr3.4.1.1
Cisco	ATA122	mr3.4.1.1
Cisco	SPA232D	mr3.4.1.1
Cisco	SPA301	mr3.4.1.1
Cisco	SPA303	mr3.4.1.1
Cisco	SPA501G	mr3.4.1.1
Cisco	SPA502G	mr3.4.1.1
Cisco	SPA512G	mr3.4.1.1
Cisco	SPA504G	mr3.4.1.1
Cisco	SPA504G + SPA500S	mr3.7.1.4
Cisco	SPA504G + two SPA500S	mr3.7.1.4
Cisco	SPA514G	mr3.4.1.1
Cisco	SPA508G	mr3.4.1.1
Cisco	SPA509G	mr3.4.1.1
Cisco	SPA525G	mr3.4.1.1
Grandstream	HT814	mr5.1.1.1

<b>Vendor</b>	<b>Model</b>	<b>Available from release</b>
Grandstream	GXW-4008	mr5.1.1.1
Grandstream	GXW-4216	mr5.1.1.1
Innovaphone	IP2X2X	mr3.8.3.3
Innovaphone	IP230-X	mr3.8.3.3
Innovaphone	IP232	mr3.8.3.3
Innovaphone	IP222	mr3.8.3.3
Innovaphone	IP240	mr3.8.3.3
Innovaphone	IP22	mr3.8.3.3
Innovaphone	IP111	mr3.8.3.3
Panasonic	KX-UT113	mr3.7.1.1
Panasonic	KX-UT123	mr3.7.1.1
Panasonic	KX-UT133	mr3.7.1.1
Panasonic	KX-UT136	mr3.7.1.1
Panasonic	KX-UT248	mr3.7.1.1
Panasonic	KX-TGP600	mr5.1.1.1
Panasonic	KX-HDV330	mr5.1.1.1
Panasonic	KX-HDV230	mr5.1.1.1
Panasonic	KX-HDV130	mr5.1.1.1
Polycom	VVX300	mr5.4.1.1
Polycom	VVX400	mr5.4.1.1
Polycom	VVX500	mr5.4.1.1
Yealink	CP860	mr5.2.1.1
Yealink	SIP-T19P	mr3.7.1.1
Yealink	SIP-T20P	mr3.7.1.1
Yealink	SIP-T21P	mr3.7.1.1
Yealink	SIP-T22P	mr3.7.1.1
Yealink	SIP-T23P	mr3.7.1.1
Yealink	SIP-T23G	mr3.7.1.1
Yealink	SIP-T26P	mr3.7.1.1
Yealink	SIP-T28P	mr3.7.1.1
Yealink	SIP-T32G	mr3.7.1.1
Yealink	SIP-T38G	mr3.7.1.1
Yealink	SIP-T41P	mr3.7.1.1
Yealink	SIP-T42G	mr3.7.1.1
Yealink	SIP-T46G	mr3.7.1.1
Yealink	SIP-T48G	mr3.7.1.1
Yealink	SIP-T28P + EXP39	mr3.8.1.1
Yealink	SIP-T28P + two EXP39	mr3.8.1.1
Yealink	W52P	mr3.7.1.6

### 16.1.13.1 Audiocodes Devices

Model	IPv6	TLS	SRTP	Auto provisioning	Private Line	Shared Line	Busy Lamp	Speed Dial
Mediant800	Y	Y	Y	dhcp	1	0	0	N

### 16.1.13.2 Cisco Devices

#### IP Phones

Model	IPv6	TLS	SRTP	Auto provisioning	Private Line	Shared Line	Busy Lamp	Extension Boards
SPA301	N	Y	Y	http	1	1	0	N
SPA303	N	Y	Y	http	1-3	1-3	1-2	N
SPA501G	N	Y	Y	http	1-8	1-8	1-7	N
SPA502G	N	Y	Y	http	1	1	0	N
SPA512G	N	N	Y	http	1	1	0	N
SPA504G	N	Y	Y	http	1-4	1-4	1-3	2
SPA514G	N	N	Y	http	1-4	1-4	1-3	N
SPA508G	N	Y	Y	http	1-8	1-8	1-7	N
SPA509G	N	Y	Y	http	1-12	1-12	1-11	N
SPA525G	N	Y	N	http	1-5	1-5	1-4	N

#### Analog Adapters

Model	IPv6	TLS	SRTP	Auto provisioning	Private Line	Shared Line	Busy Lamp
SPA232D	N	Y	Y	http	1-6	0	0
ATA112	Y	Y	Y	http	1-2	0	0
ATA122	Y	Y	Y	http	1-2	0	0

#### Extension Boards

Model	Ports	Buttons	Busy Lamp	Supported phones
SPA500S	2	32	1-32	SPA500

### 16.1.13.3 Grandstream Devices

#### Analog Adapters

Model	IPv6	TLS	SRTP	Auto provisioning	Private Line	Shared Line	Busy Lamp
HT814	N	Y	Y	redirect	4	N	N
GXW-4008	N	Y	Y	redirect	8	N	N

Model	IPv6	TLS	SRTP	Auto provisioning	Private Line	Shared Line	Busy Lamp
GXW-4216	N	Y	Y	redirect	16	N	N

#### 16.1.13.4 Innovaphone Devices

##### IP Phones

Model	IPv6	TLS	SRTP	Auto provisioning	Private Line	Shared Line	Busy Lamp	Extension Boards
IP232	N	Y	Y	dhcp	1	0	1-16	2
IP222	N	Y	Y	dhcp	1	0	1-16	2
IP240	N	N	N	dhcp	1	0	1-15	2
IP111	N	Y	Y	dhcp	1	0	1-16	0

##### Analog Adapters

Model	IPv6	TLS	SRTP	Auto provisioning	Private Line	Shared Line	Busy Lamp
IP22	N	Y	Y	dhcp	1	0	0

##### Extension Boards

Model	Ports	Buttons	Busy Lamp	Supported phones
IP2X2X	2	64	1-32	IP2x2
IP230-X	2	30	1-30	IP230

#### 16.1.13.5 Panasonic Devices

##### IP Phones

Model	IPv6	TLS	SRTP	Auto provisioning	Private Line	Shared Line	Busy Lamp	Extension Boards
KX-UT113	N	N	N	redirect	1-2	1-2	0	N
KX-UT123	N	N	N	redirect	1-2	1-2	0	N
KX-UT133	N	N	N	redirect	1-4	1-4	1-23	N
KX-UT136	N	N	N	redirect	1-4	1-4	1-23	N
KX-UT248	N	N	Y	redirect	1-6	1-6	1-23	N
KX-TGP600	Y	Y	Y	redirect	1-8	N	N	N
KX-HDV330	Y	Y	Y	redirect	1-12	Y	Y	N
KX-HDV230	Y	Y	Y	redirect	1-6	Y	Y	N
KX-HDV130	Y	Y	Y	redirect	1-2	Y	Y	N



**16.1.13.6 Polycom Devices****IP Phones**

Model	IPv6	TLS	SRTP	Auto provisioning	Private Line	Shared Line	Busy Lamp	Extension Boards
VVX300	N	N	Y	redirect	1-6	1-6	Y	N
VVX400	N	N	Y	redirect	1-12	1-12	Y	N
VVX500	N	N	Y	redirect	1-12	1-12	Y	N

**16.1.13.7 Yealink Devices****IP Phones**

Model	IPv6	TLS	SRTP	Auto provisioning	Private Line	Shared Line	Busy Lamp	Extension Boards
CP860	Y	Y	Y	redirect	1	N	N	N
SIP-T19P	Y	Y	Y	redirect	1	1	0	N
SIP-T20P	Y	Y	Y	redirect	1	1	0	N
SIP-T21P	Y	Y	Y	redirect	1-2	1-2	1	N
SIP-T22P	Y	Y	Y	redirect	1-3	1-3	1-2	N
SIP-T23P	Y	Y	Y	redirect	1-3	1-3	1-2	N
SIP-T23G	Y	Y	Y	redirect	1-3	1-3	1-2	N
SIP-T26P	Y	Y	Y	redirect	1-3	1-3	1-12	N
SIP-T28P	Y	Y	Y	redirect	1-6	1-6	1-15	2
SIP-T32G	Y	Y	Y	redirect	1-3	1-3	1-2	N
SIP-T38G	Y	Y	Y	redirect	1-6	1-6	1-15	N
SIP-T41P	Y	Y	Y	redirect	1-3	1-3	1-14	N
SIP-T42G	Y	Y	Y	redirect	1-3	1-3	1-14	N
SIP-T46G	Y	Y	Y	redirect	1-6	1-6	1-26	N
SIP-T48G	Y	Y	Y	redirect	1-6	1-6	1-28	N
W52P	N	Y	Y	redirect	1-5	1-5	0	N

**16.1.14 Phone features****16.1.14.1 Cisco phones****SPA301****1) Soft keys**

Not available.

**2) Hard keys**

- vm
- hold/unhold

### 3) Line keys

Not available.

### 4) VSC

- directed pickup
- park/unpark

## SPA303

### 1) Soft keys

#### Idle:

redial	lcr	dir	dnd >
< cfwd	unpark		

#### Idle with missed calls:

lcr			miss
-----	--	--	------

#### Call:

hold	endCall	conf	xfer >
< bxfer	park		

#### Call on hold:

resume	endCall	newCall	redial >
< dir	cfwd	dnd	

#### Ringling:

answer	ignore		
--------	--------	--	--

### 2) Hard keys

- vm
- hold/unhold

**3) Line keys**

- BLF monitoring
- directed pickup

**4) VSC**

- directed pickup

**SPA501G****1) Soft keys****Idle:**

redial	lcr	dir	dnd >
< cfwd	unpark		

**Idle with missed calls:**

lcr			miss
-----	--	--	------

**Call:**

hold/resume	endCall	conf	xfer >
< bxfer	park		

**Call on hold:**

resume	endCall	newCall	redial >
< dir	cfwd	dnd	

**Ringling:**

answer	reject		
--------	--------	--	--

**2) Hard keys**

- vm
- hold/unhold

**3) Line keys**

- BLF monitoring
- directed pickup

#### 4) VSC

- directed pickup

### SPA502G

#### 1) Soft keys

##### Idle:

redial	lcr	dir	dnd >
< cfwd	unpark		

##### Idle with missed calls:

lcr			miss
-----	--	--	------

##### Call:

hold/resume	endCall	conf	xfer >
< bxfer	park		

##### Call on hold:

resume	endCall	newCall	redial >
< dir	cfwd	dnd	

##### Ringling:

answer	reject		
--------	--------	--	--

#### 2) Hard keys

- vm
- hold/unhold

#### 3) Line keys

Not available.

#### 4) VSC

- directed pickup

### SPA504G

#### 1) Soft keys

##### Idle:

redial	lcr	dir	dnd >
< cfwd	unpark		

##### Idle with missed calls:

lcr			miss
-----	--	--	------

##### Call:

hold/resume	endCall	conf	xfer >
< bxfer	park		

##### Call on hold:

resume	endCall	newCall	redial >
< dir	cfwd	dnd	

##### Ringling:

answer	reject		
--------	--------	--	--

#### 2) Hard keys

- vm
- hold/unhold

#### 3) Line keys

- BLF monitoring
- directed pickup

#### 4) VSC

- directed pickup

**SPA512G****1) Soft keys****Idle:**

redial	lcr	dir	dnd >
< cfwd	unpark		

**Idle with missed calls:**

lcr			miss
-----	--	--	------

**Call:**

hold/resume	endCall	conf	xfer >
< bxfer	park		

**Call on hold:**

resume	endCall	newCall	redial >
< dir	cfwd	dnd	

**Ringing:**

answer	reject		
--------	--------	--	--

**2) Hard keys**

- vm
- hold/unhold

**3) Line keys**

Not available.

**4) VSC**

- directed pickup

**SPA514G****1) Soft keys****Idle:**

redial	lcr	dir	dnd >
< cfwd	unpark		

**Idle with missed calls:**

lcr			miss
-----	--	--	------

**Call:**

hold/resume	endCall	conf	xfer >
< bxfer	park		

**Call on hold:**

resume	endCall	newCall	redial >
< dir	cfwd	dnd	

**Ringling:**

answer	reject		
--------	--------	--	--

**2) Hard keys**

- vm
- hold/unhold

**3) Line keys**

- BLF monitoring
- directed pickup

**4) VSC**

- directed pickup

**SPA509G****1) Soft keys****Idle:**

redial	lcr	dir	dnd >
< cfwd	unpark		

**Idle with missed calls:**

lcr			miss
-----	--	--	------

**Call:**

hold/resume	endCall	conf	xfer >
< bxfer	park		

**Call on hold:**

resume	endCall	newCall	redial >
< dir	cfwd	dnd	

**Ringling:**

answer	reject		
--------	--------	--	--

**2) Hard keys**

- vm
- hold/unhold

**3) Line keys**

- BLF monitoring
- directed pickup

**4) VSC**

- directed pickup

**SPA508G****1) Soft keys****Idle:**

redial	lcr	dir	dnd >
< cfwd	unpark		

**Idle with missed calls:**



lcr			miss
-----	--	--	------

**Call:**

hold/resume	endCall	conf	xfer >
< bxfer	park		

**Call on hold:**

resume	endCall	newCall	redial >
< dir	cfwd	dnd	

**Ringling:**

answer	reject		
--------	--------	--	--

**2) Hard keys**

- vm
- hold/unhold

**3) Line keys**

- BLF monitoring
- directed pickup

**4) VSC**

- directed pickup

**SPA525G****1) Soft keys****Idle:**

Redial	call Rtn	Directory	DND >
< Forward	Unpark		

**Idle with missed calls:**

Call Rtn			Miss
----------	--	--	------

**Call:**

Hold	End Call	Conf	Transfer >
BlindXfer	Park		

**Call on hold:**

Resume	EndCall	EewCall	Redial >
< Directory	Forward	DND	

**Ringing:**

Answer	Ignore		
--------	--------	--	--

**2) Hard keys**

- vm
- hold/unhold

**3) Line keys**

- BLF monitoring
- directed pickup

**4) VSC**

- directed pickup

**16.1.14.2 Yealink phones****T19P****1) Soft keys****Idle:**

History		DND	Menu
---------	--	-----	------

**Idle with missed calls:**

Exit			View
------	--	--	------

**Call:**

Tran	Hold	Conf	Cancel
------	------	------	--------

**Call on hold:**

Tran	Resume	NewCall	Cancel
------	--------	---------	--------

**Ringing:**

Answer	FWD	Silence	Reject
--------	-----	---------	--------

**2) Hard keys**

- vm
- redial
- transfer

**3) Line keys**

Not available.

**4) VSC**

- transfer park
- directed pick up
- park/unpark

**T20P****1) Soft keys****Idle:**

History		DND	Menu
---------	--	-----	------

**Idle with missed calls:**

Exit			View
------	--	--	------

**Call:**

Tran	Hold	Conf	Cancel
------	------	------	--------

**Call on hold:**

Tran	Resume	NewCall	Cancel
------	--------	---------	--------

**Ringling:**

Answer	FWD	Silence	Reject
--------	-----	---------	--------

**2) Hard keys**

- vm
- redial
- transfer

**3) Line keys**

- BLF monitoring
- directed pickup

**4) VSC**

- transfer park
- park/unpark

**T21P****1) Soft keys****Idle:**

History		DND	Menu
---------	--	-----	------

**Idle with missed calls:**

Exit			View
------	--	--	------

**Call:**

Tran	Hold	Conf	Cancel
------	------	------	--------

**Call on hold:**

Tran	Resume	NewCall	Cancel
------	--------	---------	--------

**Ringling:**

Answer	FWD	Silence	Reject
--------	-----	---------	--------

**2) Hard keys**

- vm
- redial
- transfer

**3) Line keys**

- BLF monitoring
- directed pickup

**4) VSC**

- transfer park
- park/unpark

**T22P****1) Soft keys****Idle:**

History		DND	Menu
---------	--	-----	------

**Idle with missed calls:**

Exit			View
------	--	--	------

**Call:**

Tran	Hold	Conf	Cancel
------	------	------	--------

**Call on hold:**

Tran	Resume	NewCall	Cancel
------	--------	---------	--------

**Ringling:**

Answer	FWD	Silence	Reject
--------	-----	---------	--------

**2) Hard keys**

- vm
- redial
- transfer

**3) Line keys**

- BLF monitoring
- directed pickup

**4) VSC**

- park/unpark
- transfer park

**T23P****1) Soft keys****Idle:**

History		DND	Menu
---------	--	-----	------

**Idle with missed calls:**

Exit			View
------	--	--	------

**Call:**

Tran	Hold	Conf	Cancel
------	------	------	--------

**Call on hold:**

Tran	Resume	NewCall	Cancel
------	--------	---------	--------

**Ringling:**

Answer	FWD	Silence	Reject
--------	-----	---------	--------

## 2) Hard keys

- vm
- redial
- transfer

## 3) Line keys

- BLF monitoring
- directed pickup

## 4) VSC

- park/unpark
- transfer park

## T23G

### 1) Soft keys

#### Idle:

History	Dir	DND	Menu
---------	-----	-----	------

#### Idle with missed calls:

Exit			View
------	--	--	------

#### Call:

Tran	Hold	Conf	EndCall
------	------	------	---------

#### Call on hold:

Tran	Resume	NewCall	EndCall
------	--------	---------	---------

#### Ringling:

Answer	FWD		Reject
--------	-----	--	--------

**2) Hard keys**

- vm
- redial
- transfer

**3) Line keys**

- BLF monitoring
- directed pickup

**4) VSC**

- unpark
- transfer park

**T26P****1) Soft keys****Idle:**

History		DND	Menu
---------	--	-----	------

**Idle with missed calls:**

Exit			View
------	--	--	------

**Call:**

Tran	Hold	Conf	Cancel
------	------	------	--------

**Call on hold:**

Tran	Resume	NewCall	Cancel
------	--------	---------	--------

**Ringling:**

Answer	FWD	Silence	Reject
--------	-----	---------	--------

**2) Hard keys**



- vm
- redial
- transfer

### 3) Line keys

- BLF monitoring
- directed pickup

### 4) VSC

- unpark
- transfer park

### *T28P*

#### 1) Soft keys

##### Idle:

History		DND	Menu
---------	--	-----	------

##### Idle with missed calls:

Exit			View
------	--	--	------

##### Call:

Tran	Hold	Conf	Cancel
------	------	------	--------

##### Call on hold:

Tran	Resume	NewCall	Cancel
------	--------	---------	--------

##### Ringling:

Answer	FWD	Silence	Reject
--------	-----	---------	--------

#### 2) Hard keys

- vm
- redial

- transfer

### 3) Line keys

- BLF monitoring
- directed pickup

### 4) VSC

- park/unpark
- transfer park

## T32G

### 1) Soft keys

#### Idle:

History		DND	Menu
---------	--	-----	------

#### Idle with missed calls:

Exit			View
------	--	--	------

#### Call:

Tran	Hold	Conf	Cancel
------	------	------	--------

#### Call on hold:

Tran	Resume	NewCall	Cancel
------	--------	---------	--------

#### Ringling:

Answer	FWD	Silence	Reject
--------	-----	---------	--------

### 2) Hard keys

- vm
- redial
- transfer

**3) Line keys**

- BLF monitoring
- directed pickup

**4) VSC**

- unpark
- transfer park

**T38G****1) Soft keys****Idle:**

History		DND	Menu
---------	--	-----	------

**Idle with missed calls:**

Exit			View
------	--	--	------

**Call:**

Tran	Hold	Conf	Cancel
------	------	------	--------

**Call on hold:**

Tran	Resume	NewCall	Cancel
------	--------	---------	--------

**Ringling:**

Answer	FWD	Silence	Reject
--------	-----	---------	--------

**2) Hard keys**

- vm
- redial
- transfer

**3) Line keys**

- BLF monitoring
- directed pickup

#### 4) VSC

- unpark
- transfer park

### **T41P**

#### 1) Soft keys

##### Idle:

History		DND	Menu
---------	--	-----	------

##### Idle with missed calls:

Exit			View
------	--	--	------

##### Call:

Tran	Hold	Conf	Cancel
------	------	------	--------

##### Call on hold:

Tran	Resume	NewCall	Cancel
------	--------	---------	--------

##### Ringling:

Answer	FWD	Silence	Reject
--------	-----	---------	--------

#### 2) Hard keys

- vm
- redial
- transfer

#### 3) Line keys

- BLF monitoring
- directed pickup

**4) VSC**

- park/unpark
- transfer park

**T42G****1) Soft keys****Idle:**

History		DND	Menu
---------	--	-----	------

**Idle with missed calls:**

Exit			View
------	--	--	------

**Call:**

Tran	Hold	Conf	Cancel
------	------	------	--------

**Call on hold:**

Tran	Resume	NewCall	Cancel
------	--------	---------	--------

**Ringling:**

Answer	FWD	Silence	Reject
--------	-----	---------	--------

**2) Hard keys**

- vm
- redial
- transfer

**3) Line keys**

- BLF monitoring
- directed pickup

**4) VSC**

- park/unpark
- transfer park

## T46G

### 1) Soft keys

#### Idle:

History		DND	Menu
---------	--	-----	------

#### Idle with missed calls:

Exit			View
------	--	--	------

#### Call:

Tran	Hold	Conf	Cancel
------	------	------	--------

#### Call on hold:

Tran	Resume	NewCall	Cancel
------	--------	---------	--------

#### Ringling:

Answer	FWD	Silence	Reject
--------	-----	---------	--------

### 2) Hard keys

- vm
- redial
- transfer

### 3) Line keys

- BLF monitoring
- directed pickup

### 4) VSC

- park/unpark
- transfer park

**T48G****1) Soft keys****Idle:**

History		DND	Menu
---------	--	-----	------

**Idle with missed calls:**

Exit			View
------	--	--	------

**Call:**

Tran	Hold	Conf	Cancel
------	------	------	--------

**Call on hold:**

Tran	Resume	NewCall	Cancel
------	--------	---------	--------

**Ringling:**

Answer	FWD	Silence	Reject
--------	-----	---------	--------

**2) Hard keys**

- vm
- redial
- transfer

**3) Line keys**

- BLF monitoring
- directed pickup

**4) VSC**

- park/unpark
- transfer park

**W52P****1) Soft keys**

**Idle:**



History	Line
---------	------

**Idle with missed calls:**

Exit	View
------	------

**Call:**

Ext. Call	Options
-----------	---------

**Call on hold:**

Resume	Line
--------	------

**Ringing:**

Accept	
--------	--

**2) Hard keys**

- vm
- redirect

**3) VSC**

- park/unpark
- transfer park

**16.1.14.3 Panasonic phones*****KX-UT113*****1) Soft keys****Idle:**

Settings	Call Log	Phone book	
----------	----------	------------	--

**Call:**

Blind		Phone book	
-------	--	------------	--

**Call on hold:**

	Call Log	Phone book	
--	----------	------------	--

**Ringling:**

Answer		Reject	
--------	--	--------	--

**2) Hard keys**

- vm
- forward/dnd
- hold/unhold
- redial
- recall
- transfer
- conf

**3) Line keys**

Not available.

**4) VSC**

- park/unpark
- transfer park

***KX-UT123*****1) Soft keys****Idle:**

Settings	Call Log	Phone book	
----------	----------	------------	--

**Call:**

Blind		Phone book	
-------	--	------------	--

**Call on hold:**

	Call Log	Phone book	
--	----------	------------	--

**Ringling:**

Answer		Reject	
--------	--	--------	--

**2) Hard keys**

- vm
- forward/dnd
- hold/unhold
- redial
- recall
- transfer
- conf

**3) Line keys**

Not available.

**4) VSC**

- park/unpark
- transfer park

***KX-UT133*****1) Soft keys****Idle:**

Settings	Call Log	Phone book	
----------	----------	------------	--

**Call:**

Blind		Phone book	
-------	--	------------	--

**Call on hold:**

	Call Log	Phone book	
--	----------	------------	--

**Ringling:**

Answer		Reject	
--------	--	--------	--

**2) Hard keys**

- vm
- forward/dnd
- hold/unhold
- redial
- recall
- transfer
- conf

**3) Line keys**

- BLF monitoring
- directed pickup

**4) VSC**

- unpark
- transfer park

***KX-UT136*****1) Soft keys****Idle:**

Settings	Call Log	Phone book	
----------	----------	------------	--

**Call:**

Blind		Phone book	
-------	--	------------	--

**Call on hold:**

	Call Log	Phone book	
--	----------	------------	--

**Ringling:**

Answer		Reject	
--------	--	--------	--

**2) Hard keys**

- vm
- forward/dnd
- hold/unhold
- redial
- recall
- transfer
- conf

**3) Line keys**

- BLF monitoring
- directed pickup

**4) VSC**

- park/unpark
- transfer park

***KX-UT248*****1) Soft keys****Idle:**

Settings	Call Log	Phone book	
----------	----------	------------	--

**Call:**

Blind		Phone book	
-------	--	------------	--

**Call on hold:**

	Call Log	Phone book	
--	----------	------------	--

**Ringling:**

Answer		Reject	
--------	--	--------	--

**2) Hard keys**

- vm
- forward/dnd
- hold/unhold
- redial
- recall
- transfer
- conf

**3) Line keys**

- BLF monitoring
- directed pickup

**4) VSC**

- park/unpark
- transfer park

**16.1.14.4 Innovaphone****IP222****1) Soft keys****Idle:**

Setup	All Calls	Home	Calls	My favorites	Phonebook
-------	-----------	------	-------	--------------	-----------

**Call:**

Hold	Transfer	Park	Cancel
------	----------	------	--------

**Call on hold:**

Resume	Transfer	Park	Cancel
--------	----------	------	--------

**Ringling:**

Answer	Transfer	Silence	Reject
--------	----------	---------	--------

**2) Hard keys**

- hold
- redial

**3) Line keys**

- BLF monitoring

**4) VSC**

- unpark
- transfer park

***IP232*****1) Soft keys****Idle:**

Setup	All Calls	Home	Calls	My favorites	Phonebook
-------	-----------	------	-------	--------------	-----------

**Call:**

Hold	Transfer	Park	Cancel
------	----------	------	--------

**Call on hold:**

Resume	Transfer	Park	Cancel
--------	----------	------	--------

**Ringling:**

Answer	Transfer	Silence	Reject
--------	----------	---------	--------

**2) Hard keys**

- hold
- redial

**3) Line keys**

- BLF monitoring

#### 4) VSC

- unpark
- transfer park

### **IP111**

#### 1) Soft keys

##### Idle:

Setup	All Calls	Home	Calls	My favorites	Phonebook
-------	-----------	------	-------	--------------	-----------

##### Call:

Hold	Transfer	Park	Cancel
------	----------	------	--------

##### Call on hold:

Resume	Transfer	Park	Cancel
--------	----------	------	--------

##### Ringling:

Answer	Transfer	Silence	Reject
--------	----------	---------	--------

#### 2) Hard keys

- hold
- redial

#### 3) Line keys

- BLF monitoring

#### 4) VSC

- unpark
- transfer park

### **IP240**



### 1) Soft keys

Not available.

### 2) Hard keys

- hold
- redial
- conference
- dnd
- forward

### 3) Line keys

- BLF monitoring

### 4) VSC

- transfer park
- unpark

#### 16.1.15 Shared line appearance

In PBX environment, shared line appearance is supported for PBX subscribers. In comparison to the private line, subscriber registering for the shared line will, immediately after the successful registration, subscribe for Call-Info event. This subscribe is challenged for authentication and if the credentials are provided, subscriber is notified that the subscription is active. In the respective NOTIFY message, this is reflected in Subscription-State header set to active. NOTIFY also contains information about the status of the shared line in Call-Info header. If the appearance is not used, Call-Info header will describe the state as idle. In the NOTIFY message, this is reflected as “appearance-index=\*;appearance-state=idle”.

If there is incoming call to the subscriber, the appearance index is created after the call is accepted and state will be set to active. Call-Info header will contain “appearance-index=1; appearance-state=active”. After call is finished and appearance is not used elsewhere, appearance index is removed and state is set to idle. In the case of outgoing call, subscription to line-seize event is required to be able to dial. Before dialing can be started, SUBSCRIBE to line-seize is sent. Consequently, subscriber receives NOTIFY for line-seize with active subscription state. Call-info subscription is updated accordingly, appearance is created and its state is set to seized. As soon as the call starts ringing, Call-Info status is updated to progressing and line-seize subscription is set to terminated with “reason=noresource” in Subscription-State header. When the call is accepted, Call-Info status is changed to active and set again to idle when call is finished. Also, the appearance index is removed.

## 16.2 Sipwise sip:phone App (SIP client)

You can order two commercial Unified Communication Clients for full end-to-end integration of voice, video, chat and presence features. There are two applications available:

- the sip:phone Desktop Client for Microsoft Windows, Apple OSX, and Linux;
- the sip:phone Mobile App for iOS and Android.

Both clients are fully brandable to the customer's corporate identity. The clients are not part of the standard delivery and need to be licensed separately. This handbook discusses the mobile client in details.

We continuously develop the mobile clients to provide new features, as they do not support the full range of features yet.

The sip:phone Mobile App is a mobile client for iOS and Android that supports voice calls via SIP, as well as presence and instant messaging via XMPP. The following sections describe the steps needed to integrate it into your sip:carrier.

### 16.2.1 Zero Config Launcher

Part of the mobile apps is a mechanism to sign up to the service via a 3rd party website, which is initiated on the login screen and rendered within the app. During the sign-up process, the 3rd party service is supposed to create a new account and subscriber in the sip:carrier (e.g. automatically via the API) and provide the end user with the access credentials.

The mobile apps come with a zero config mechanism to simplify the end-customer log in using these credentials (especially ruling out the need to manually enter them). It makes it possible to deliver the access credentials via a side channel (e.g. Email, SMS) packed into a URL. The user just clicks the URL, and it automatically launches the app with the correct credentials. The following picture shows the overall workflow.

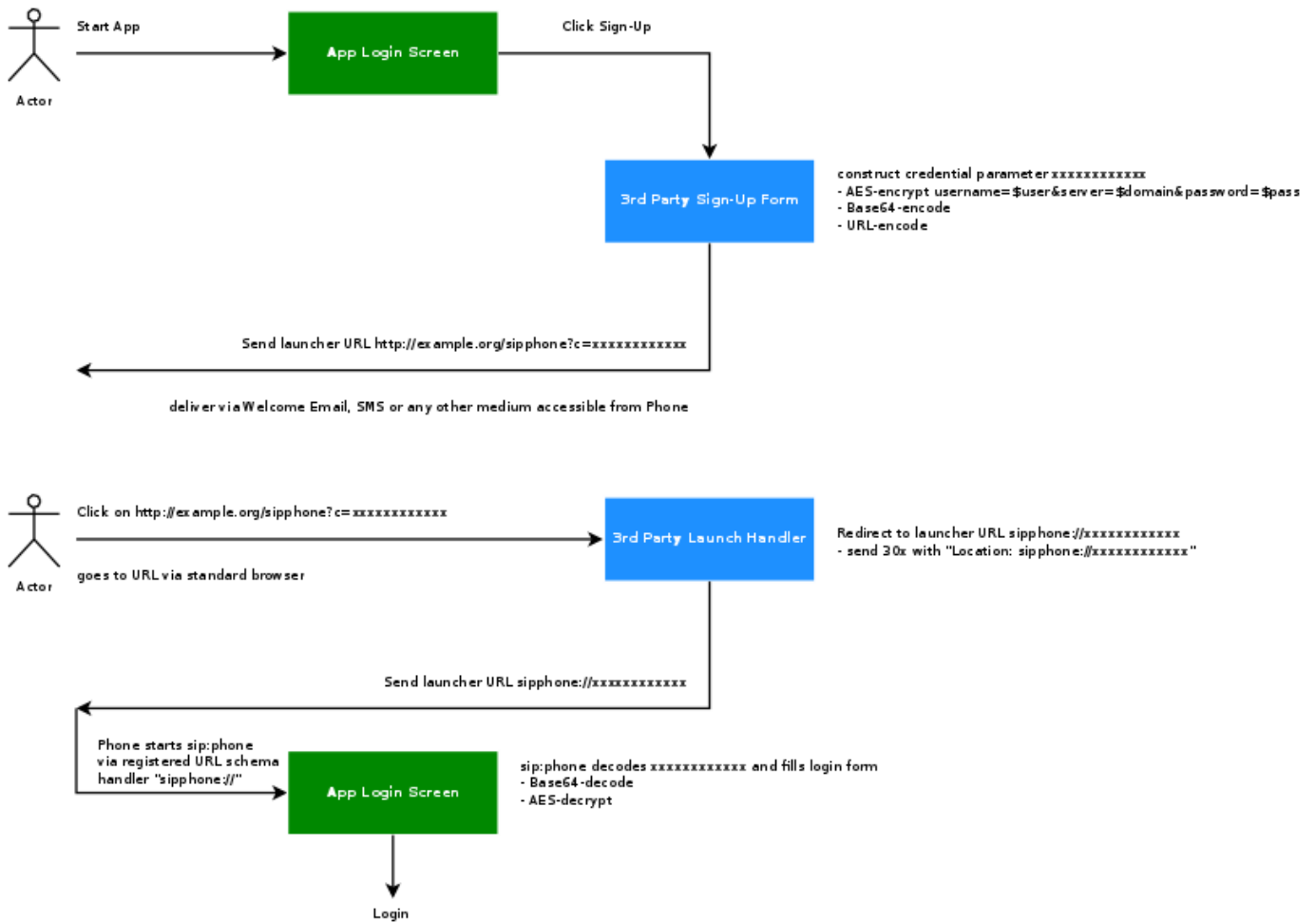


Figure 144: Provisioning Push Workflow

There are two components provided by a 3rd party system. One is the *3rd Party Sign-Up Form*, and the other is the *3rd Party Launch Handler*. The purpose of these components is to allow an end customer to open a link with the access credentials via the sip:phone app.

### 16.2.1.1 3rd Party Sign-Up Form

The 3rd Party Sign-Up Form is a website the app shows to the end user when he taps the sign-up link on the app *Login Screen*. There, the end customer usually provides his contact details like name, address, phone number and email address, etc. After validation, the website creates an account and a subscriber in the sip:carrier via the API.

After successfully creating the account and the subscriber, this site needs to construct a specially crafted URL, which is sent back to the end customer via a side channel. Ideally, this channel would be an SMS if you want to verify the end customer’s mobile number, or an email if you want to check the email address.

The sip:phone app registers a URL schema handler for URLs starting with `sipphone://`. If you start such a link, the app performs a Base64 decoding of the string right after the `sipphone://` prefix and then decrypts the resulting binary string via AES using the keys defined during the branding step. The resulting string is supposed to be

`username=$user&server=$domain&password=$password`.

Therefore, the *3rd Party Sign-Up Form* needs to construct this string using the credentials defined while creating the subscriber via the sip:carrier API, then encrypt it via AES, and finally perform a Base64 encoding of the result.

---

### Note

Up until and including version mr6.1.2 of the sip:carrier, the SIP login credentials are used here. Future versions will connect to the REST interface of the sip:carrier using the web credentials first and fetch the SIP credentials along with other settings from there.

---

An example Perl code performs encoding of such a string. The AES key and initialization vector (`$key` and `$iv`) are the standard values of the sip:phone app and should work until you specified other values during the branding process.

```
#!/usr/bin/perl -w
use strict;
use Crypt::Rijndael;
use MIME::Base64;
use URI::Escape;

my $key = 'iBmTdavJ8joPW3HO';
my $iv = 'tww211Qe6cmywrp3';

my $plain = do { local $/; <> };
# pkcs#5 padding to 16 bytes blocksize
my $pad = 16 - (length $plain) % 16;
$plain .= pack('C', $pad) x $pad;

my $cipher = Crypt::Rijndael->new(
    $key,
    Crypt::Rijndael::MODE_CBC()
);
$cipher->set_iv($iv);
my $crypt = $cipher->encrypt($plain);
# store b64-encoded string and print to STDOUT
my $b64 = encode_base64($crypt, '');
print $b64, "\n";
# print to STDOUT using URL escaping also
print uri_escape($b64), "\n";
```

This snippet takes a string from STDIN, encrypts it via AES, encodes it via Base64 and sends the result to STDOUT. It also writes the second line with the same string, but this time, the URL is escaped. To test it, you would run it as follows on a shell, granted it's stored at `/path/to/encrypt.pl`.

```
echo -n 'username=testuser&server=example.org&password=testpass' \
| /path/to/encrypt.pl
```

This command would result in the output strings `CI8VN8toaE40w8E4OH2rAuFj3Qev9QdLI/Wv/VaBCVK2yNkBZjxE9`

eafXkkrQfmYdeu01PquS5P40zhUq8Mfjg== and CI8VN8toaE40w8E4OH2rAuFj3Qev9QdLI%2FWv%2FVaBCVK2yNkBZjxE9eafXkkrQfmYdeu01PquS5P40zhUq8Mfjg%3D%3D. The sip:phone can use the former string to automatically fill in the login form of the Login Screen if started via a Link like sipphone://CI8VN8toaE40w8E4OH2rAuFj3Qev9QdLI/Wv/VaBCVK2yNkBZjxE9eafXkkrQfmYdeu01PquS5P40zhUq8Mfjg==.

Here is the same code in PHP.

```
#!/usr/bin/php
<?php
$key = "iBmTдавJ8joPW3H0";
$iv = "twW21lQe6cmYwrp3";

$clear = fgets(STDIN);
$cipher = fnEncrypt($clear, $key, $iv);

echo $cipher, "\n";
echo urlencode($cipher), "\n";

function fnEncrypt($clear, $key, $iv) {
    $pad = 16 - strlen($clear) % 16;
    $clear .= str_repeat(pack('C', $pad), $pad);
    return rtrim(base64_encode(mcrypt_encrypt(
        MCRYPT_RIJNDAEL_128, $key, $clear,
        MCRYPT_MODE_CBC, $iv)), "\0");
}
?>
```

Similar to the Perl code, you can call it like this:

```
echo -n 'username=testuser&server=example.org&password=testpass' \
| /path/to/encrypt.php
```

However, a URL with the sipphone:// schema is not displayed as a link in an SMS or an Email client and thus can not be clicked by the end customer, so you need to make a detour via a regular http:// URL. To do so, you need a *3rd Party Launch Handler* to trick the phone to open such a link.

Therefore, that the *3rd Party Sign-Up Form* needs to return a link containing a URL pointing to the *3rd Party Launch Handler* and pass the URL escaped string gathered above to the client via an SMS or an Email. Since it is the regular http:// link, it is clickable on the phone and can be launched from virtually any client (SMS, Email, etc.), which correctly renders an HTML link.

A possible SMS sent to the end customer (via the phone number entered in the sign-up form) could, therefore, look as follows (trying to stay below 140 chars).

```
http://example.org/p?c=CI8VN8toaE40w8E4OH2rAuFj3Qev9QdLI
%2FWv%2FVaBCVK2yNkBZjxE9eafXkkrQfmYdeu01PquS5P40zhUq8Mfjg%3D%3D to launch sipphone
```

An HTML Email could look like this:

```
Welcome to Example.org,
```

```
<a href="http://www.example.org/sipphone?c=CI8VN8toaE40w8E40H2rAuFj3Qev9QdLI
%2FWv%2FVaBCVK2yNkBZjxE9eafXkkrQfmYdeu01PquS5P40zhUq8Mfjg%3D%3D">
click here
</a> to log in.
```

That way, you can do both: verify the contact details of the end customer, and send the end customer the login credentials in a secure manner.

### 16.2.1.2 3rd Party Launch Handler

The URL `http://www.example.org/sipphone` mentioned above can be any simple script, and its sole purpose is to send back a 301 Moved Permanently or 302 Moved Temporarily with a `Location: sipphone://xxxxxxx xxxxxx` header to tell the phone to open this link via the sip:phone app. The `xxxxxxxxxxx` is the plain (non-URL-escaped) string generated by the above script.

An example CGI script performing this task follows.

```
#!/usr/bin/perl -w
use strict;
use CGI;

my $q = CGI->new;
my $c = $q->param('c');
print CGI::redirect("sipphone://$c");
```

The script simply takes the URL parameter `c` from the URL `http://www.example.org/sipphone?c=CI8VN8toaE40w8E40H2rAuFj3Qev9QdLI%2FWv%2FVaBCVK2yNkBZjxE9eafXkkrQfmYdeu01PquS5P40zhUq8Mfjg%3D%3D` crafted above and puts its content into a `Location` header using the `sipphone://` schema, and finally sends a 301 Moved Permanently back to the phone.

The phone follows the redirect by opening the URL using the sip:phone app, which in turn decrypts the content and fills in the login form.

---

#### Note

Future versions of the sip:carrier will be shipped with this launch handler integrated into the system. Up until and including the version mr6.1.2, this script needs to be installed on any webserver manually.

---

### 16.2.2 Mobile Push Notification

The *mobile push* functionality provides the remote start of a mobile application on incoming calls via the Google GCM or the Apple APNS notification services. It enables you to offer your subscribers a modern and convenient service on mobile devices.



**Caution**

Although suspending an application on a phone and waking it up via the mobile push notification service extends battery life, the whole mobile push notification concept is the best effort framework provided by Apple and Google for iOS and Android respectively, and therefore does not guarantee 100% reliability.

**16.2.2.1 Architecture**

If the *mobile push* functionality is enabled and there are no devices registered for a subscriber, the call-flow looks as follows.

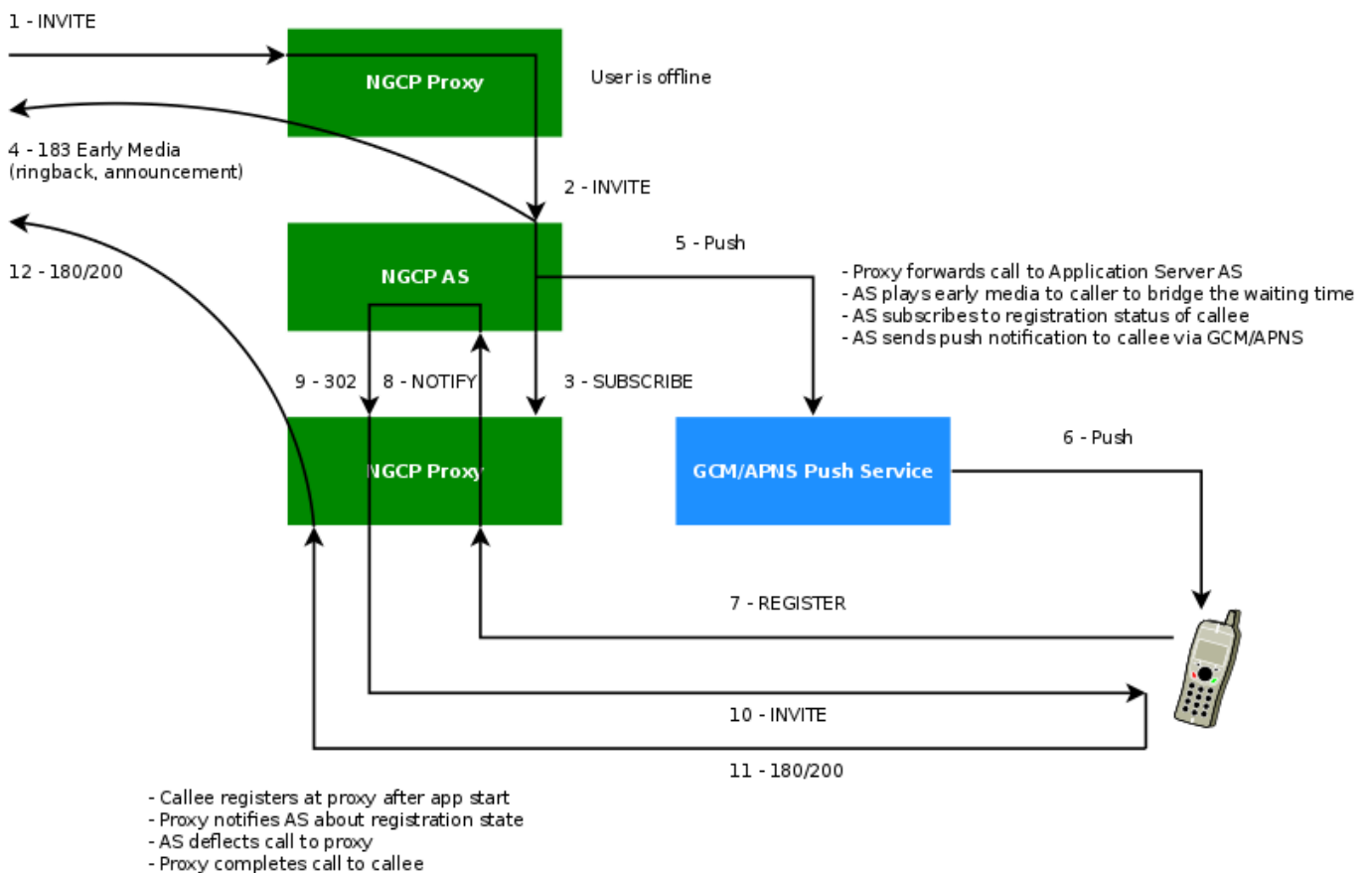


Figure 145: Mobile Push Workflow

1. The caller sends INVITE to proxy
2. The callee is offline, proxy forwards the call to AS (application server)
3. AS subscribes to the callee's registration events on proxy
4. AS sends early media to the caller as a feedback, as the call initiation process might take a while
5. AS sends the push request to GCM/APNS service
6. GCM/APNS service delivers the push request to the callee

7. The callee accepts the push request and confirms the mobile application start (unattended on Android), then the mobile application registers to proxy
8. Proxy sends registration notification to AS
9. AS deflects the call back to proxy
10. Proxy sends INVITE to the callee
11. The callee accepts the call
12. The response is sent back to the caller. Hence, the call setup is completed

In the case of a time-out (no registration notification within a particular time), the application server rejects the call request with an error.

### 16.2.2.2 The Configuration Checklist

Follow this checklist to make sure you've completed all the steps. If you miss anything, the service may not work as expected.

Name	Description	Link
Obtain a trusted SSL certificate from a CA	Required for either application	Section <a href="#">16.2.2.3</a>
Create an Apple developer account and enable the push notification service	For iOS mobile application	Section <a href="#">16.2.2.4</a>
Obtain the Apple certificate for the app	For iOS mobile application	Section <a href="#">16.2.2.5</a>
Obtain the API key for the app from Google	For Android mobile application	Section <a href="#">16.2.2.6</a>
Provide the required information to developers	It is required to make beta builds and publish the apps	Section <a href="#">16.2.2.7</a>
Adjust the configuration	Adjust the config.yml file and apply the changes (usually performed by Sipwise)	Section <a href="#">16.2.2.8</a>
Recheck your DNS Zone configuration	Check that the DNS Zone is correctly configured	Section <a href="#">16.2.2.9</a>
Add DNS SRV records	Create specific DNS SRV records for SIP and XMPP services	Section <a href="#">16.2.2.10</a>
Check NTP configuration	Ensure that all your servers show exact time	Section <a href="#">16.2.2.11</a>
Enable Apple/Google Mobile Push in the Admin Panel	It can be enabled for a domain or separate subscribers	Section <a href="#">16.2.2.12</a>
Configure a mobile application	Check that subscribers can easily install and use your application	Section <a href="#">16.2.2.13</a>



### 16.2.2.3 Obtain the Trusted SSL Certificate

A *trusted* SSL certificate is required, and we suggest obtaining it before starting the configuration.

The mobile application uses respective iOS/Android libraries to establish a secure TLS connection with certain sip:carrier services, such as SIP/XMPP/pushd(https). A *signed* SSL certificate is required to guarantee the security of this connection.

Any Certificate Authority (CA) such as Verisign and others can provide you with the required trusted SSL certificate (a certificate and the key files) which you will use in the configuration below.

### 16.2.2.4 Create an Apple Account and Enable the Push Notification Service

Below is a brief instruction on how to create an Apple account and enable the Push Notification Service in it. You may need to perform additional steps depending on your project.

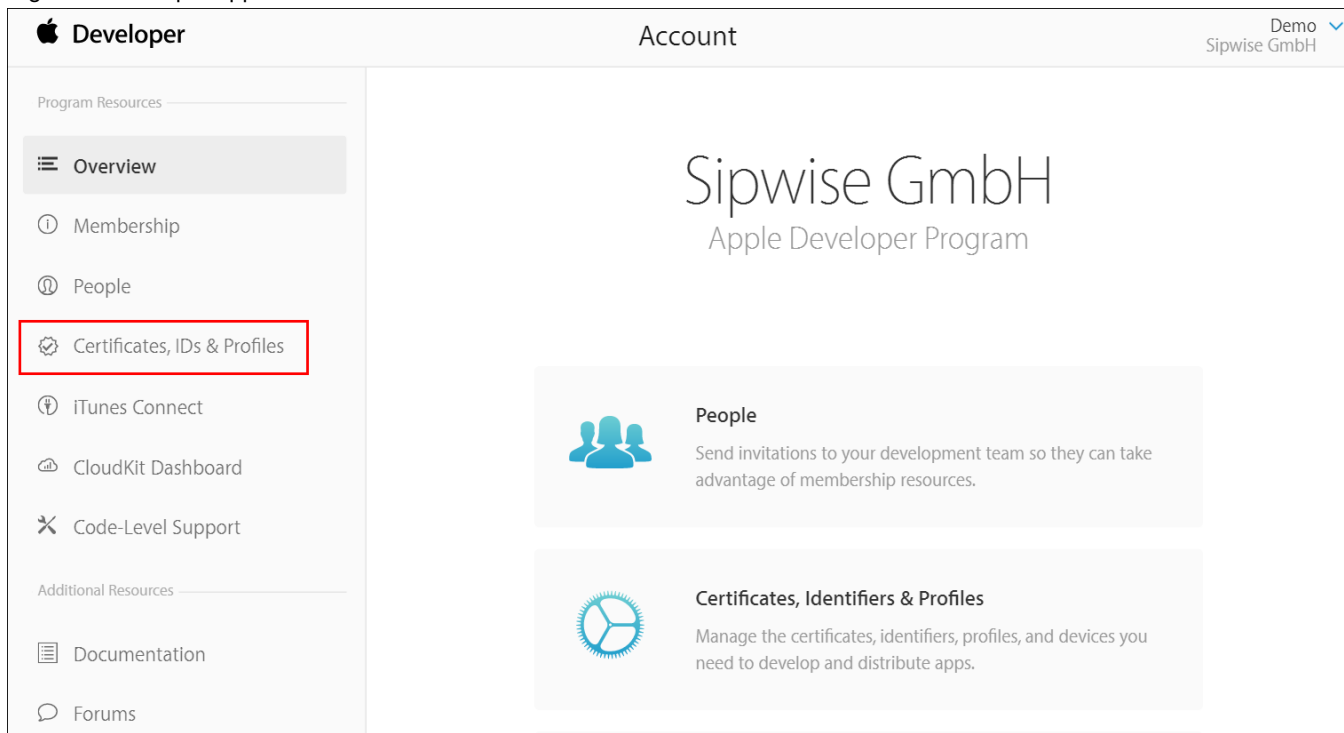
---

#### Note

You may only create an Apple account (step 1 below) and enroll into the Apple Developer Program (step 2 below) and Sipwise developers will do the rest. Still, you can perform all the steps by yourself.

---

1. Create an Apple developer account to get the Apple ID for your company. For this, go to [developer.apple.com/account](https://developer.apple.com/account)
2. Enrol in the Apple Developer Program. It is required to configure push notifications as you will need a push notification certificate for your App ID, which requires the Apple Developer Program membership. Go to [developer.apple.com/programs](https://developer.apple.com/programs) for more details.
3. Register an App ID:
  - Sign into [developer.apple.com/account](https://developer.apple.com/account).

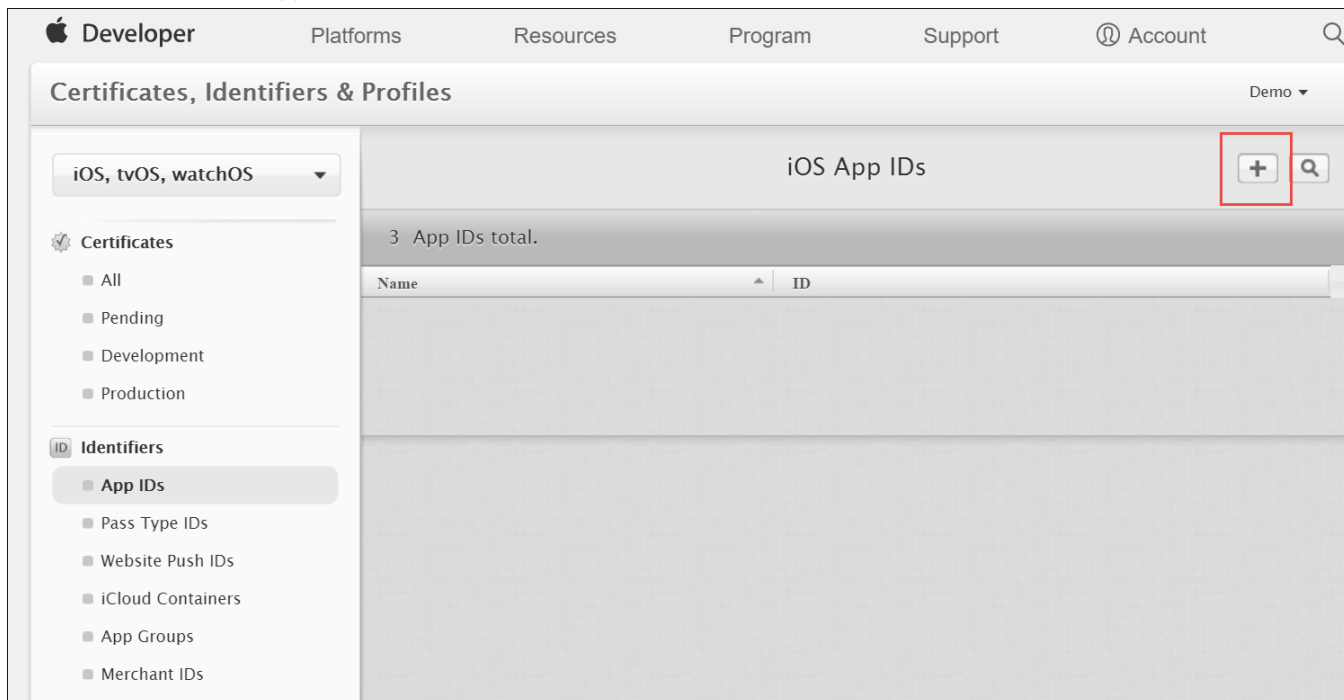


The screenshot shows the Apple Developer account interface. The top navigation bar includes the Apple logo, the word 'Developer', and the account name 'Sipwise GmbH' with a 'Demo' indicator. The left sidebar lists various program resources, with 'Certificates, IDs & Profiles' highlighted by a red box. The main content area displays the company name 'Sipwise GmbH' and the 'Apple Developer Program' logo. Below this, there are two main sections: 'People' and 'Certificates, Identifiers & Profiles'. The 'Certificates, Identifiers & Profiles' section is currently selected and highlighted.

- Click *Certificates, IDs & Profiles*.



- Under *Identifiers*, select *App IDs*.



- Click the *Add* button (+) in the upper-right corner.

ID

## Registering an App ID

---

The App ID string contains two parts separated by a period (.) — an App ID Prefix that is defined as your Team ID by default and an App ID Suffix that is defined as a Bundle ID search string. Each part of an App ID has different and important uses for your app. [Learn More](#)

---

### App ID Description

Name:

You cannot use special characters such as @, &, \*, ', "

- Enter a name for the App ID in the *App ID Description* block. This helps you identify the App ID later.

- Apple TV
- Apple Watch
- iPad
- iPhone
- iPod Touch

---

Provisioning Profiles

- All
- Development
- Distribution

### App ID Prefix

Value: XD7GAT4I26 (Team ID)

---

### App ID Suffix

**Explicit App ID**

If you plan to incorporate app services such as Game Center, In-App Purchase, Data Protection, and iCloud, or want a provisioning profile unique to a single app, you must register an explicit App ID for your app.

To create an explicit App ID, enter a unique string in the Bundle ID field. This string should match the Bundle ID of your app.

Bundle ID:

We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (\*).

**Wildcard App ID**

This allows you to use a single App ID to match multiple apps. To create a wildcard App ID, enter an asterisk (\*) as the last digit in the Bundle ID field.

- Select *Explicit App ID* and enter the app's bundle ID in the *Bundle ID* field. Note that an explicit App ID exactly matches the bundle ID of an app you are building — for example, com.example.push. An explicit App ID can *not* contain an asterisk (\*).

### App Services

Select the services you would like to enable in your app. You can edit your choices after this App ID has been registered.

Enable Services:

- App Groups
- Associated Domains
- Game Center
- In-App Purchase
- Inter-App Audio
- Wallet
- Push Notifications
- Personal VPN

- In the App Services section enable Push Notifications. Click *Continue* to submit the form

### ID Confirm your App ID.

To complete the registration of this App ID, make sure your App ID information is correct, and click the submit button.

App ID Description: **com example push**

Identifier: **XD7GAT4I26**

Data Protection:  Disabled

Game Center:  **Enabled**

iCloud:  Disabled

In-App Purchase:  **Enabled**

Inter-App Audio:  Disabled

Passbook:  Disabled

Push Notifications:  **Enabled**

- Click *Submit* to create the App ID.

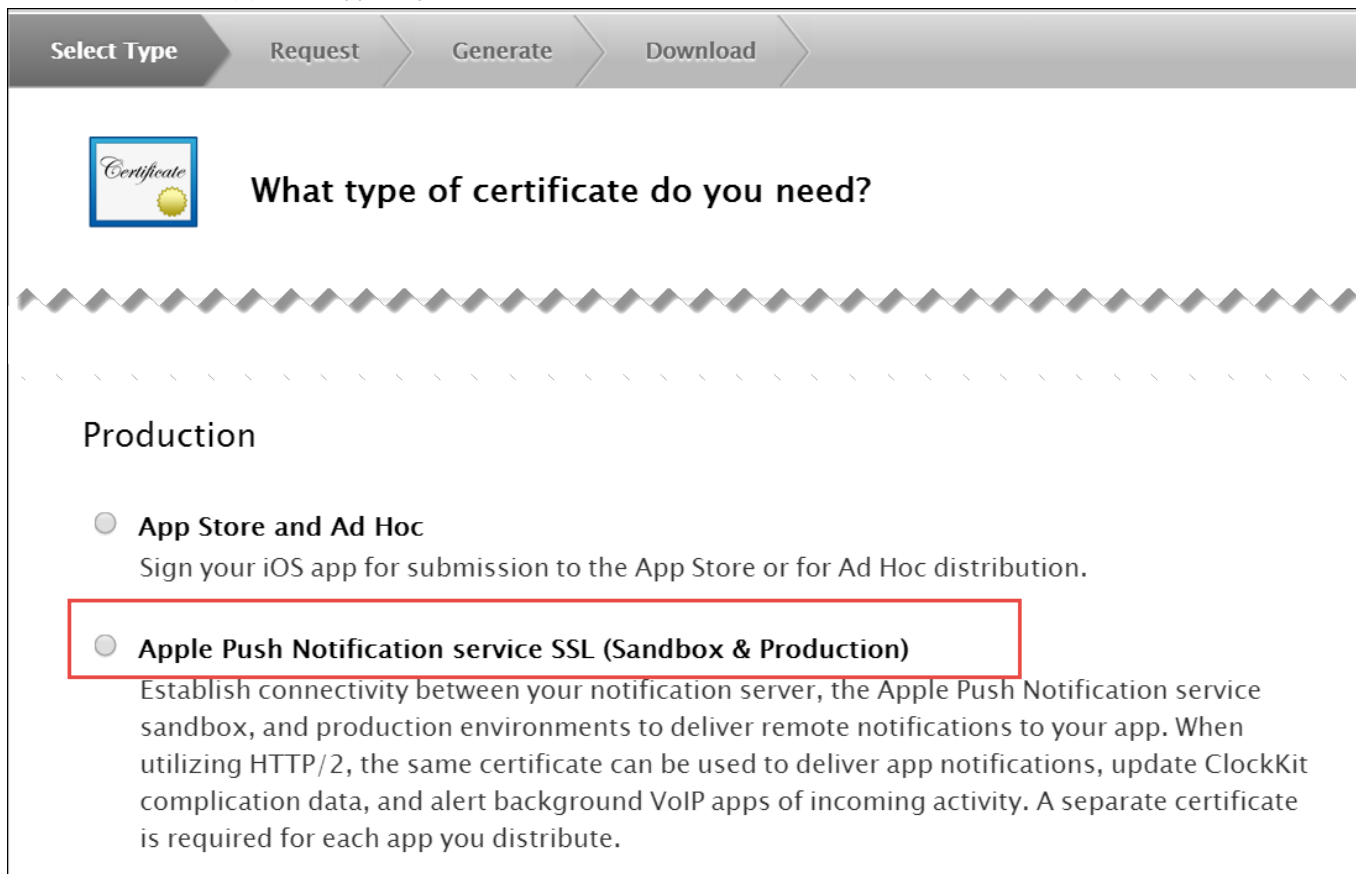
### 16.2.2.5 Obtain an Apple SSL Certificate and a Private Key

1. Create a CSR (Certificate Signing Request):

- Sign into [developer.apple.com/account/ios/certificate](https://developer.apple.com/account/ios/certificate).




- Click the *Add* button (+) in the upper-right corner.



- Select *Apple Push Notification service SSL (Sandbox & Production)* as the certificate type and click *Continue*.

Select Type   Request   Generate   Download



### Which App ID would you like to use?


All App IDs that you want to enable for remote notifications require their own Apple Push Notification service SSL certificate. The App ID-specific SSL certificate allows your server to connect to the Apple Push Notification service. Note that only explicit App IDs with a specific Bundle Identifier can be used to create an Apple Push Notification service SSL certificate.

Select an App ID for your Apple Push Notification service SSL Certificate (Sandbox & Production)

App ID:

- Select your App ID and click *Continue*.

Select Type Request Generate Download



## About Creating a Certificate Signing Request (CSR)

To manually generate a Certificate, you need a Certificate Signing Request (CSR) file from your Mac. To create a CSR file, follow the instructions below to create one using Keychain Access.

**Create a CSR file.**

In the Applications folder on your Mac, open the Utilities folder and launch Keychain Access.

Within the Keychain Access drop down menu, select Keychain Access > Certificate Assistant > Request a Certificate from a Certificate Authority.

- In the Certificate Information window, enter the following information:
  - In the User Email Address field, enter your email address.
  - In the Common Name field, create a name for your private key (e.g., John Doe Dev Key).
  - The CA Email Address field should be left empty.
  - In the "Request is" group, select the "Saved to disk" option.
- Click Continue within Keychain Access to complete the CSR generating process.

- Read the information about creating a CSR.
- Follow the instructions to create a CSR using Keychain Access in MAC.

---

**Note**


If you do not have access to a Mac, you can still create a CSR in Linux or Windows using OpenSSL, for example.

---

## 2. Get the Certificate and Private Key

- When you have the CSR file return to the browser and click *Continue*.

Select TypeRequestGenerateDownload



## Generate your certificate.

---

When your CSR file is created, a public and private key pair is automatically generated. Your private key is stored on your computer. On a Mac, it is stored in the login Keychain by default and can be viewed in the Keychain Access app under the "Keys" category. Your requested certificate is the public half of your key pair.


**Upload CSR file.**  
Select .certSigningRequest file saved on your Mac.

Choose File...

- Click *Choose File...* in your browser.



Select Type Request Generate Download



## Generate your certificate.


When your CSR file is created, a public and private key pair is automatically generated. Your private key is stored on your computer. On a Mac, it is stored in the login Keychain by default and can be viewed in the Keychain Access app under the "Keys" category. Your requested certificate is the public half of your key pair.

**Upload CSR file.**  
Select .certSigningRequest file saved on your Mac.

CertificateSigningRequest.certSigningRequest

- Select the CSR file you just created and saved and click *Continue*.


Select Type
Request
Generate
Download



## Your certificate is ready.

---

**Download, Install and Backup**  
 Download your certificate to your Mac, then double click the .cer file to install in Keychain Access. Make sure to save a backup copy of your private and public keys somewhere secure.



**Name:** Apple Push Services: com.example.push

**Type:** Apple Push Services

**Expires:** Jun 26, 2017

Download

- Click *Download* to download the certificate (give it the **aps.cer** name).
- Open the downloaded certificate file (it should automatically be opened in Keychain Access, otherwise open it manually in Keychain Access).
- Find the certificate you just opened/imported in Keychain Access.
- Expand the certificate to show the Private Key.
- Select only the Private Key portion of the certificate, right-click on it and select *Export "Common Name"...* from the menu.
- Choose a location (e.g. Desktop) and filename to export the .p12 file to and click *Save*.
- **Optionally** pick a password for the .p12 file to protect its private key contents and click *OK*. (You will then need to enter your log-in password to permit the export).

### 3. Generate a PEM file from the p12 file:

- Open up your terminal and run the following commands to create a PEM file from the p12 file (If you input a password for the p12 file, you will need to enter it here):

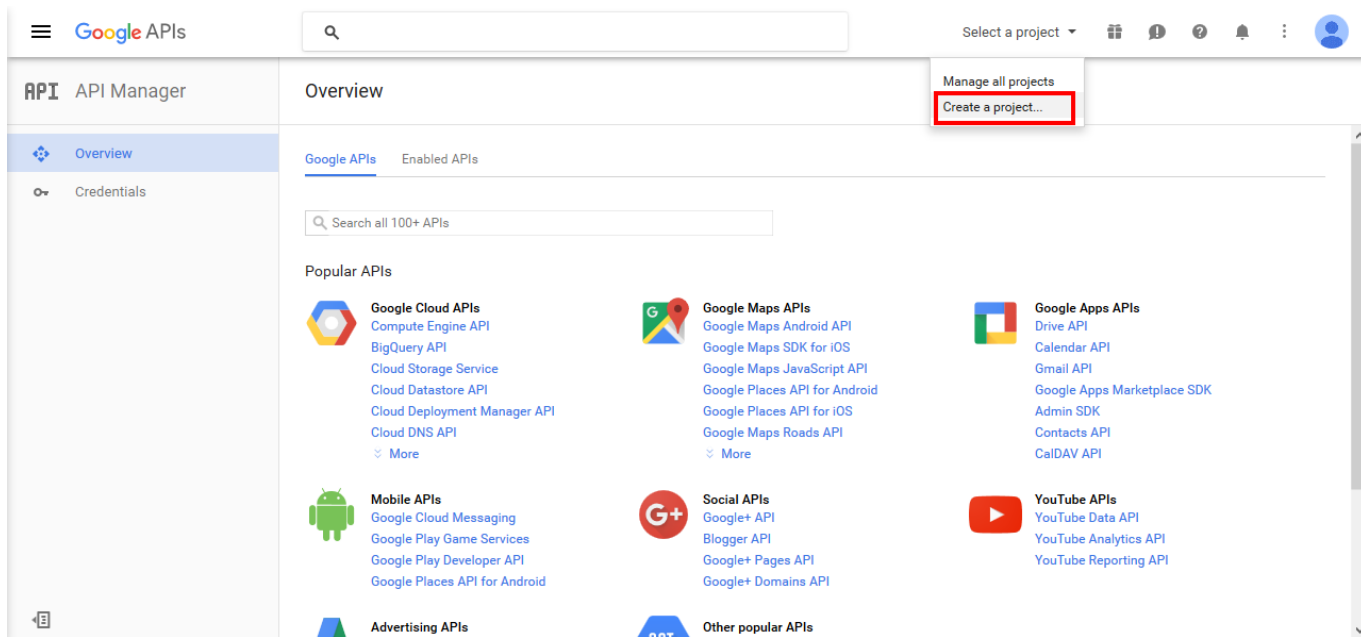
```
cd ~/Desktop
openssl x509 -in aps.cer -inform der -out PushChatCert.pem
openssl pkcs12 -in PushChatCert.p12 -out PushCertificate.pem -nodes -clcerts
openssl pkcs12 -nocerts -out PushChatKey.pem -in PushChatKey.p12
```

### 16.2.2.6 Obtain the API Key for the App from Google

You can use Google Cloud Messaging (GCM) to send push notifications to your subscribers with Android-based mobile devices. Google Cloud Messaging is a free service that acts as an intermediary between the NGCP and devices of your subscribers. Google's Cloud Connection Server (CCS), a part of GCP, manages the persistent connections with mobile devices to deliver your push notifications.

While communicating with CCS, the NGCP identifies itself using an API key. To get it, follow the steps below.

1. Create a new project in the Google APIs Console page. For this go to [code.google.com/apis/console](https://code.google.com/apis/console).



2. Click *Create a Project*.

review

le APIs    Enab

Search all 100+ A

ilar APIs

- Google Cl
- Compute
- BigQuery
- Cloud Sto
- Cloud Dat
- Cloud Dep
- Cloud DNS
- More

### New Project

**Project name** ?

Your project ID will be mychat-1320 ? [Edit](#)

[Show advanced options...](#)

**Please email me updates regarding feature announcements, performance suggestions, feedback surveys and special offers.**

Yes  No

**I agree that my use of any services and related APIs is subject to my compliance with the applicable [Terms of Service](#).**

Yes  No

[Create](#) [Cancel](#)



Mobile APIs  
Google Cloud Messaging



Social APIs  
Google+ API

3. Input the project name, agree with the *Terms of Service* and click *Create*.

**API** API Manager

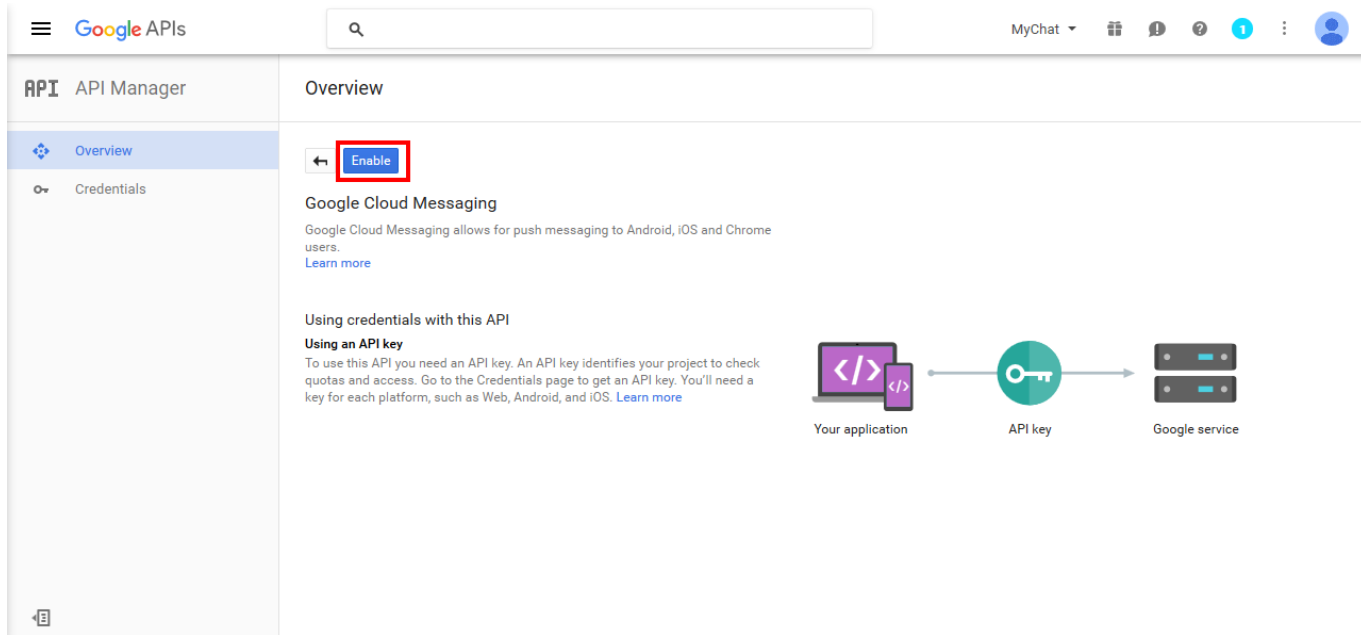
Overview

Google APIs    Enabled APIs (0)

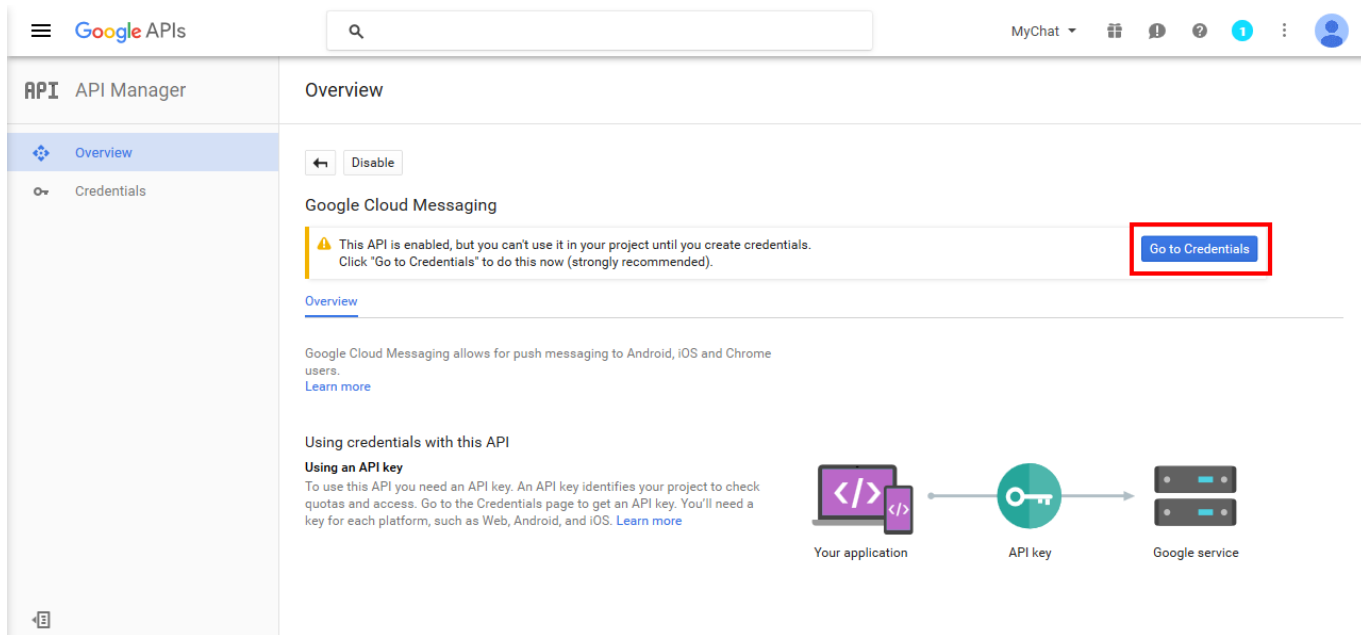
Popular APIs

- Google Cloud APIs**
  - Compute Engine API
  - BigQuery API
  - Cloud Storage Service
  - Cloud Datastore API
  - Cloud Deployment Manager API
  - Cloud DNS API
  - More
- Mobile APIs**
  - Google Cloud Messaging**
  - Google Play Game Services
  - Google Play Developer API
  - Google Places API for Android
- Google Maps APIs**
  - Google Maps Android API
  - Google Maps SDK for iOS
  - Google Maps JavaScript API
  - Google Places API for Android
  - Google Places API for iOS
  - Google Maps Roads API
  - More
- Social APIs**
  - Google+ API
  - Blogger API
  - Google+ Pages API
  - Google+ Domains API
- Google Apps APIs**
  - Drive API
  - Calendar API
  - Gmail API
  - Google Apps Marketplace SDK
  - Admin SDK
  - Contacts API
  - CalDAV API
- YouTube APIs**
  - YouTube Data API
  - YouTube Analytics API
  - YouTube Reporting API

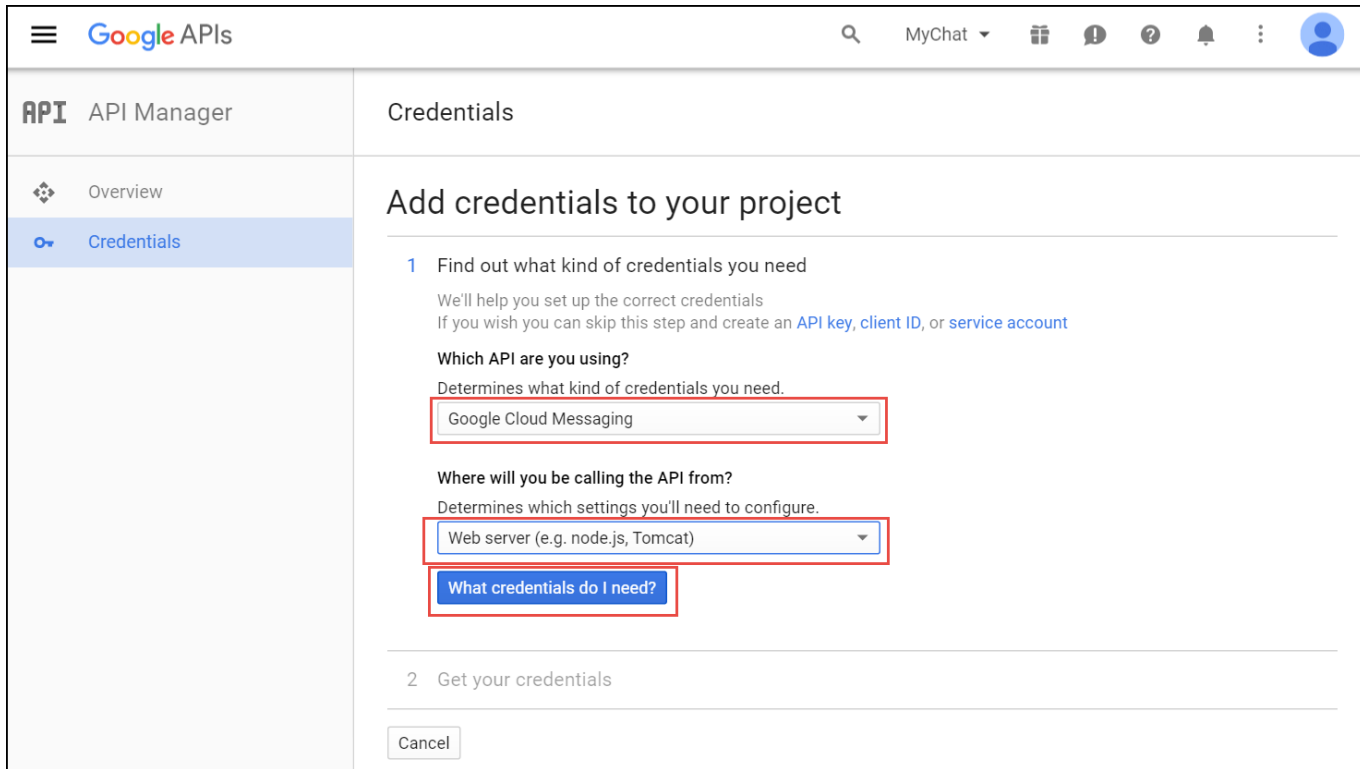
4. Click *Google Cloud Messaging* on the Overview page.



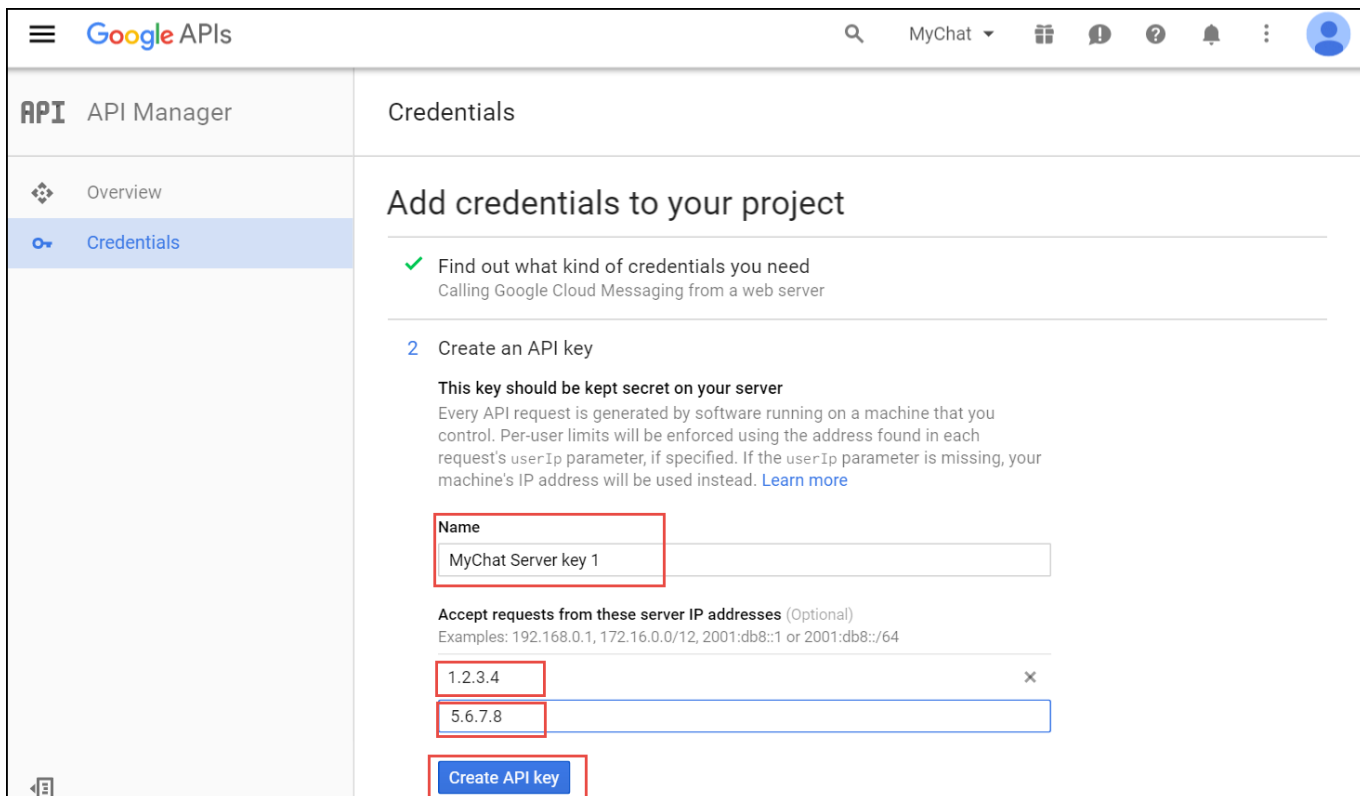
5. Click *Enable* for the Google Cloud Messaging.



6. Click *Go to Credentials*.



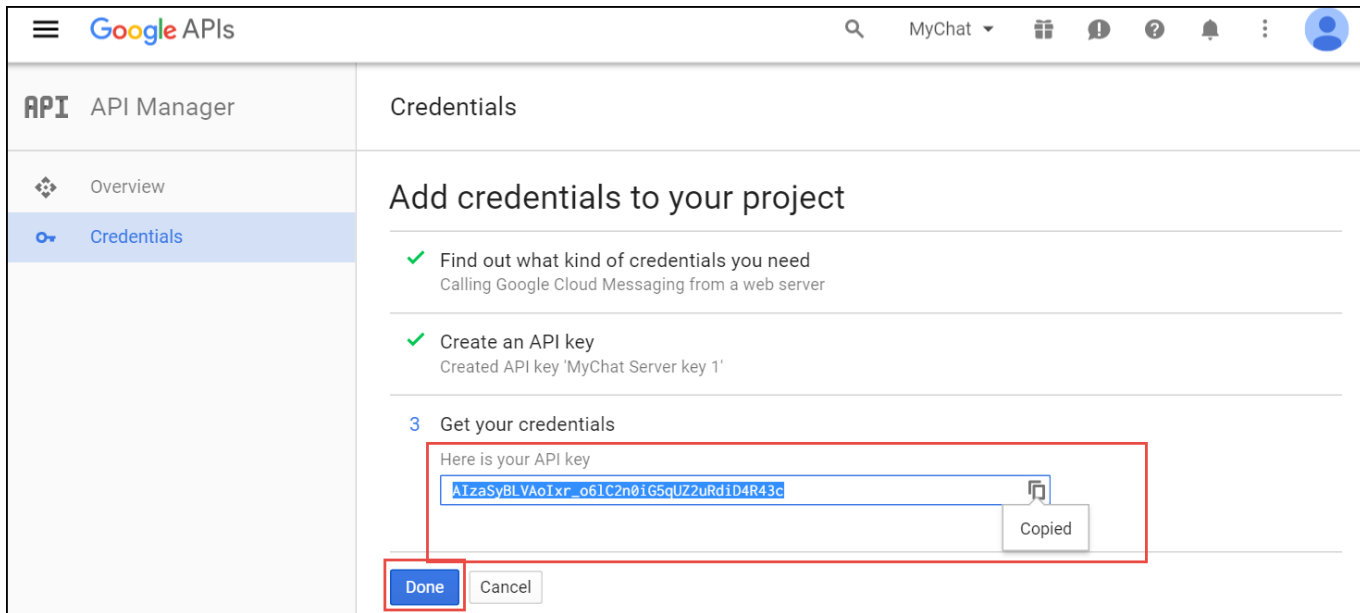
7. Select Google Cloud Messaging and Web Server from the corresponding lists and click *What credentials do I need?*



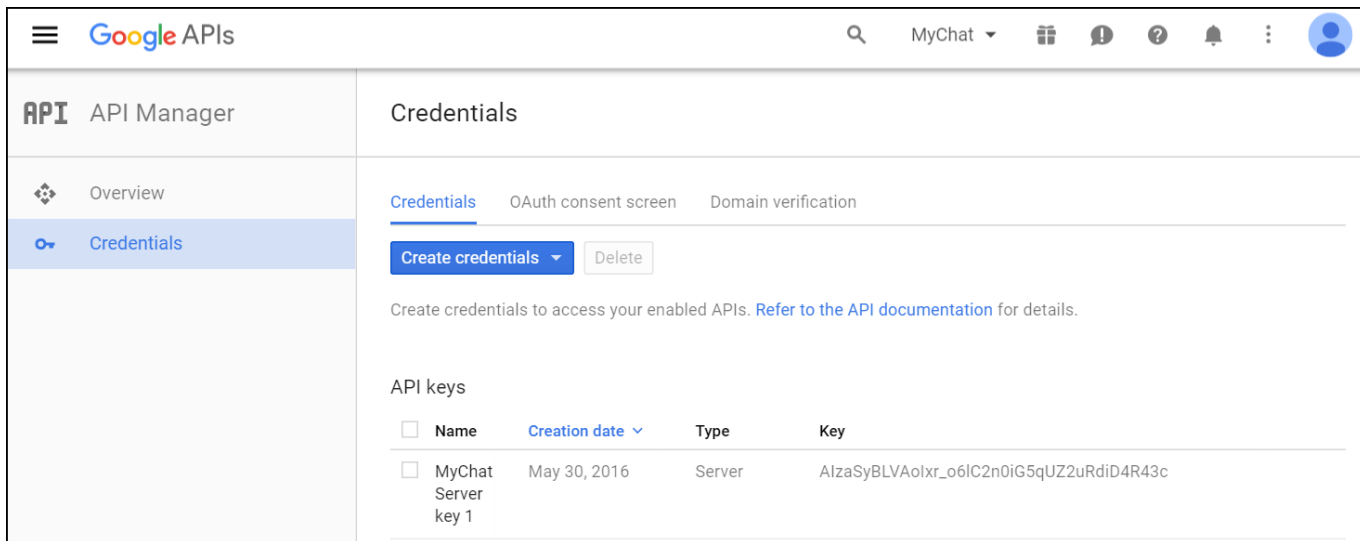
8. Adjust the API Key name and input the IP addresses of *all* your load balancers under *Accept requests from these server IP addresses*. Click *Create API key*.

**Note**

You may skip adding the IP addresses, otherwise list *ALL* your load balancers.



9. Copy your API key and click *Done*. Save the API key for future use.



**16.2.2.7 Provide the Required Information to Developers**

Please, provide Sipwise developers with the following files and information so that they can make beta builds and submit the application to the App Store:

- Access to your Apple developer account
- The trusted SSL certificate and its private key
- The Apple SSL certificate and its private key

For the Android application, provide the following:

- Access to your Google developer account
- Google application API key

### 16.2.2.8 Adjust the sip:carrier Configuration (Usually Performed by Sipwise)

1. Upload the Apple SSL certificate (**PushChatCert.pem**) and the private key (**PushChatKey.pem**) to `/etc/ngcp-config/ssl/`
2. Upload the trusted SSL certificate (**CAsigned.crt**) and the private key (**CAsigned.key**) to `/etc/ngcp-config/ssl/`
3. Specify the corresponding paths and names in the pushd section of the config.yml file:

- apns: section (For iOS mobile application)
  - certificate: `'/etc/ngcp-config/ssl/PushChatCert.pem'`
  - enable: yes
  - key: `'/etc/ngcp-config/ssl/PushChatKey.pem'`
- enable: yes
- gcm: section (for Android mobile application)
  - enable: yes
  - key: `'google_server_api_key_here'`
- ssl: yes
- sslcertfile: `/etc/ngcp-config/ssl/CAsigned.crt`
- sslcertkeyfile: `/etc/ngcp-config/ssl/CAsigned.key`

You can find an example of `/etc/ngcp-config/config.yml` configuration in the [config.yml overview section](#).

4. Apply your changes:

```
ngcpcfg apply 'enabled the backup feature.'
ngcpcfg push all
```

### 16.2.2.9 Recheck Your DNS Zone Configuration

Check that your **NS** and **A** DNS records are correctly configured.

Let's consider the following example: \* the load-balancers have the lb01a.example.com and the lb01b.example.com names \* the shared name is lb01.example.com and the shared IP address is 1.1.1.1 \* the service name is voipservice.example.com

The following DNS records must be present:

Server Name	Record type	IP Address
lb01a.example.com	A	1.2.3.4
lb01b.example.com	A	5.6.7.8
lb01.example.com	A	1.1.1.1
voipservice.example.com	A	1.1.1.1



### 16.2.2.10 Add SRV Records to DNS

Add at least one record for each service: **xmpp-server**, **xmpp-client**, **sips**.

A regular SRV record has the following form:

```
_service._proto.name. TTL class SRV priority weight port target
```

- **service**: the symbolic name of the service (xmpp-server, xmpp-client, sips).
- **proto**: the transport protocol of the desired service (TCP).
- **name**: the domain name (ending in a dot).
- **TTL**: standard DNS time to live field.
- **class**: the standard DNS class field (this is always IN).
- **priority**: the priority of the target host (lower value means more preferred).
- **weight**: a relative weight for records with the same priority (the higher the value, the more requests will be sent).
- **port**: the TCP or UDP port of the service.
- **target**: the canonical hostname of the machine providing the service (ending in a dot).

Here are examples of the SRV records:

```
_xmpp-server._tcp.voipservice.example.com. 18000 IN SRV 10 50 5269 voipservice.example.com.  
_xmpp-client._tcp.voipservice.example.com. 18000 IN SRV 10 50 5222 voipservice.example.com.  
_sips._tcp.voipservice.example.com. 18000 IN SRV 10 100 5061 voipservice.example.com.
```

You can always check whether the required SRV records are configured by executing the following commands:

```
dig SRV _xmpp-client._tcp.voipservice.example.net  
dig SRV _xmpp-server._tcp.voipservice.example.net  
dig SRV _sips._tcp.voipservice.example.net
```

### 16.2.2.11 Check NTP Configuration

We strongly suggest that the clocks of all the nodes within the platform are synchronized. To ensure this, check that the NTP service is correctly configured on all your sip:carrier servers and works reliably. Execute the following command for quick test of time synchronization:

```
ntpq -p
```

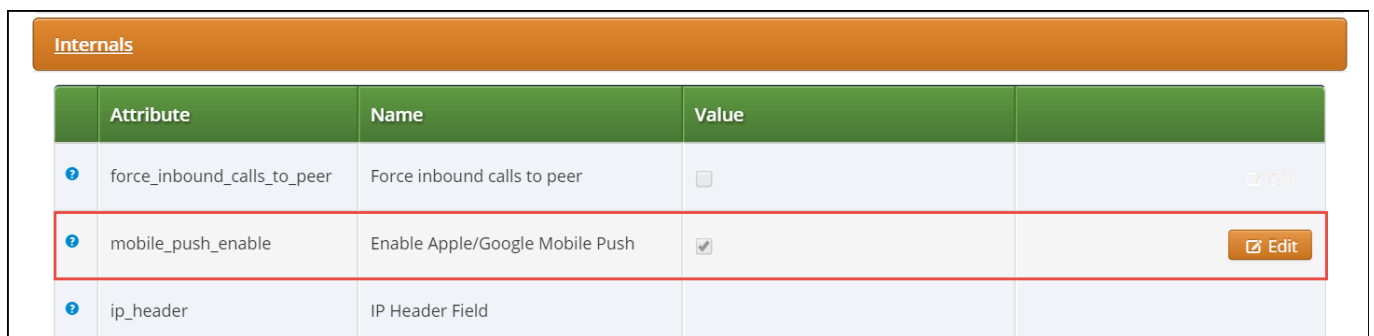
If the current node synchronizes with an NTP server, this server will be marked by the star (\*) symbol.

### 16.2.2.12 Enable Apple/Google Mobile Push

It can be enabled for a domain or separate subscribers in the Admin Panel.

To enable the service for a domain:

1. Go to *Settings*→*Domains* and click on the *Preferences* button of the domain you want to enable Apple/Google Mobile Push for.
2. Go to the *Internals* group and enable the **mobile\_push\_enable** parameter.



The screenshot shows the 'Internals' settings page. It features a table with columns for 'Attribute', 'Name', and 'Value'. The 'mobile\_push\_enable' row is highlighted with a red border, indicating it is the focus of the instruction. The 'Value' column for this row shows a checked checkbox, and an 'Edit' button is visible to the right.

Attribute	Name	Value
force_inbound_calls_to_peer	Force inbound calls to peer	<input type="checkbox"/>
mobile_push_enable	Enable Apple/Google Mobile Push	<input checked="" type="checkbox"/>
ip_header	IP Header Field	

### 16.2.2.13 Perform Tests

Perform tests when the application is available:

1. Download and install the application.
2. Open the application and input your registration username in the username@domain.name format and password.
3. Review the quality of application branding.
4. Make test calls.
5. Test the presence functionality.
6. Test the chat and group chat.
7. Test messaging.
8. Test the sharing functionality (e.g. pictures, video and voice messages and maps).
9. Check the application phone book integration with the phone's one

Make sure that the subscribers can start using your services in the easiest possible way.

### 16.3 Lawful Interception

#### 16.3.1 Introduction

The Sipwise sip:carrier, as a communications platform carrying voice, fax and messaging data has to provide means for lawful interception of the content of communication by third party entities. Those Law Enforcement Agencies (LEAs) have to be able to connect to the Sipwise NGCP platform in a standardized way — ETSI, 3GPP and other organisations define the interface (and data exchange) between telecommunication operators and LEAs.

High level overview of lawful interception is shown in the following figure:

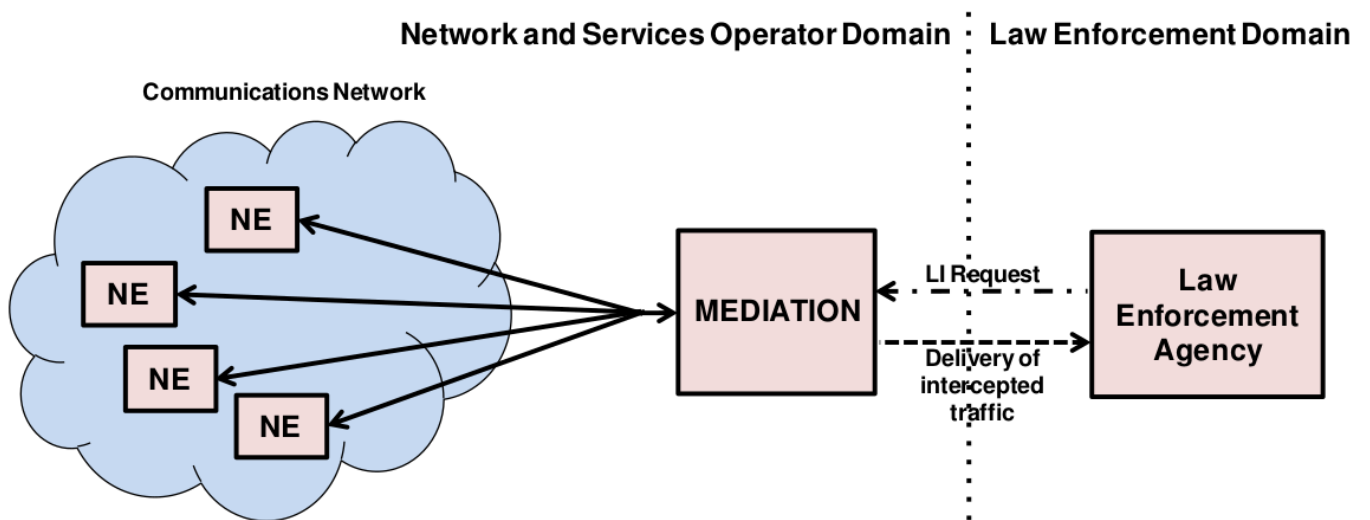


Figure 146: LI: High Level Overview

Main interfaces of lawful interception according to ETSI standard:

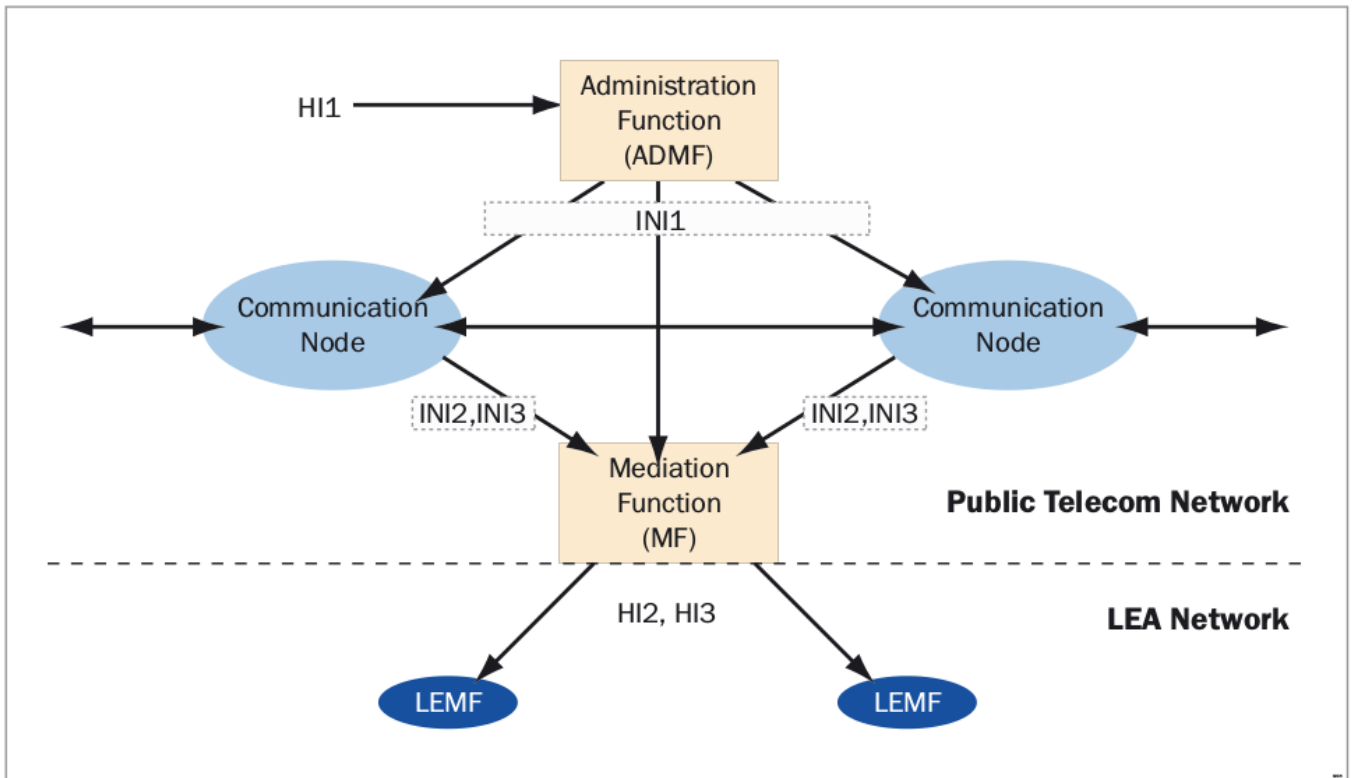


Figure 147: LI: ETSI Interfaces

**16.3.1.1 Terms and Abbreviations**

**Content of Communication (CC)**

Information exchanged between two or more users of a telecommunications service, excluding Intercept Related Information.

**Note**

This includes information which may, as part of some telecommunications service, be stored by one user for subsequent retrieval by another.

**CC Internal Interception Function (CC-IIF)**

The CC-IIF shall cause the CC, specified by the CCTF, via the CCCI to be duplicated and passed to the MF.

**Content of Communication Control Interface (CCCI)**

Carries controls information from the CCTF to the CC-IIF.

**CC Trigger Function (CCTF)**

The purpose of the CCTF is to determine the location of the CC-IIF device associated to the target CC traffic, and to control the CC-IIF via the CCCI interface.

**Content of Communication Trigger Interface (CCTI)**

Carries trigger information from the IRI-IIF to the CCTF.

**Handover Interface (HI)**

Physical and logical interface across which the interception measures are requested from an operator, and the results of interception are delivered from an operator to an LEMF.

**Intercept Related Information (IRI)**

Collection of information or data associated with telecommunication services involving the target identity, specifically call or service associated information or data (e.g. call identifier, unsuccessful call attempts) and location information.

**Intercept Related Information Internal Interception Function (IRI-IIF)**

The purpose of the IRI-IIF is to generate IRI information associated with sessions, calls, connections and any other information involving interception targets identified by Law Enforcement Agency (LEA) sessions.

**Internal Network Interface (INI)**

Network's internal interface between the Internal Intercepting Function and a mediation function.

**Law Enforcement Agency (LEA)**

Organization authorized, by a lawful authorization based on a national law, to request interception measures and to receive the results of telecommunications interceptions.

**Law Enforcement Monitoring Facility (LEMF)**

Law enforcement facility designated as the transmission destination for the results of interception relating to a particular interception subject.

**Lawful Interception Administration Function (AF)**

The AF ensures that an intercept request from a LEA for IRI or CC or both is provisioned for collection from the network, and subsequent delivery to the LEMF.

**Lawful Interception Mediation Function (MF)**

Mechanism which passes information between an access provider or network operator or service provider and a handover interface.

1. Firstly it receives information related to active intercepts from the IRI-IIF(s) and CC-IIF(s) within the service provider network.
2. Secondly correlates and formats that IRI and CC information in real time for delivery to the LEMF over the HI2 and HI3 handover Interfaces.

**X1, X2 and X3 Interfaces**

The 3GPP standard for Lawful Interception defines the handover interfaces with different names compared to the ETSI standard. The X $n$  interface corresponds to the INI $n$  interface and is functionally identical to the INI $n$  interface.

**16.3.2 Architecture and Configuration of LI Service**

Sipwise sip:carrier platform implements the functions defined by LI requirements in a way that it relies on a third party provider for the Lawful Interception Mediation Function (MF).

Regarding other LI functions that are defined by ETSI / 3GPP standards there are 2 possible implementations:

1. Sipwise NGCP behaves as the Administration Function (AF) but the actual call data capturing is carried out by other SIP endpoints. In this case NGCP forwards the calls to be intercepted to its **SIP peers dedicated for LI service**. Within the scope of SIP peer based solution there are still 2 modes of operation:
  - *Call loopback to NGCP*: the LI peer receives the call, extracts IRI and CC data and then routes the call back to NGCP. NGCP handles the looped back call as if that was initiated from NGCP and sets up the second call leg to the destination.
  - *Call forwarded by peer directly to destination*: in this case NGCP will handle the call to LI peer as an ordinary second call leg to the destination.
2. Sipwise **NGCP itself provides** the required LI functions: AF and call data capturing; IRI and CC of intercepted calls are forwarded to the third party MF from NGCP.

This handbook will discuss the second setup in details in the following sections.

The below figure illustrates the logical connection of LI functions on Sipwise NGCP.

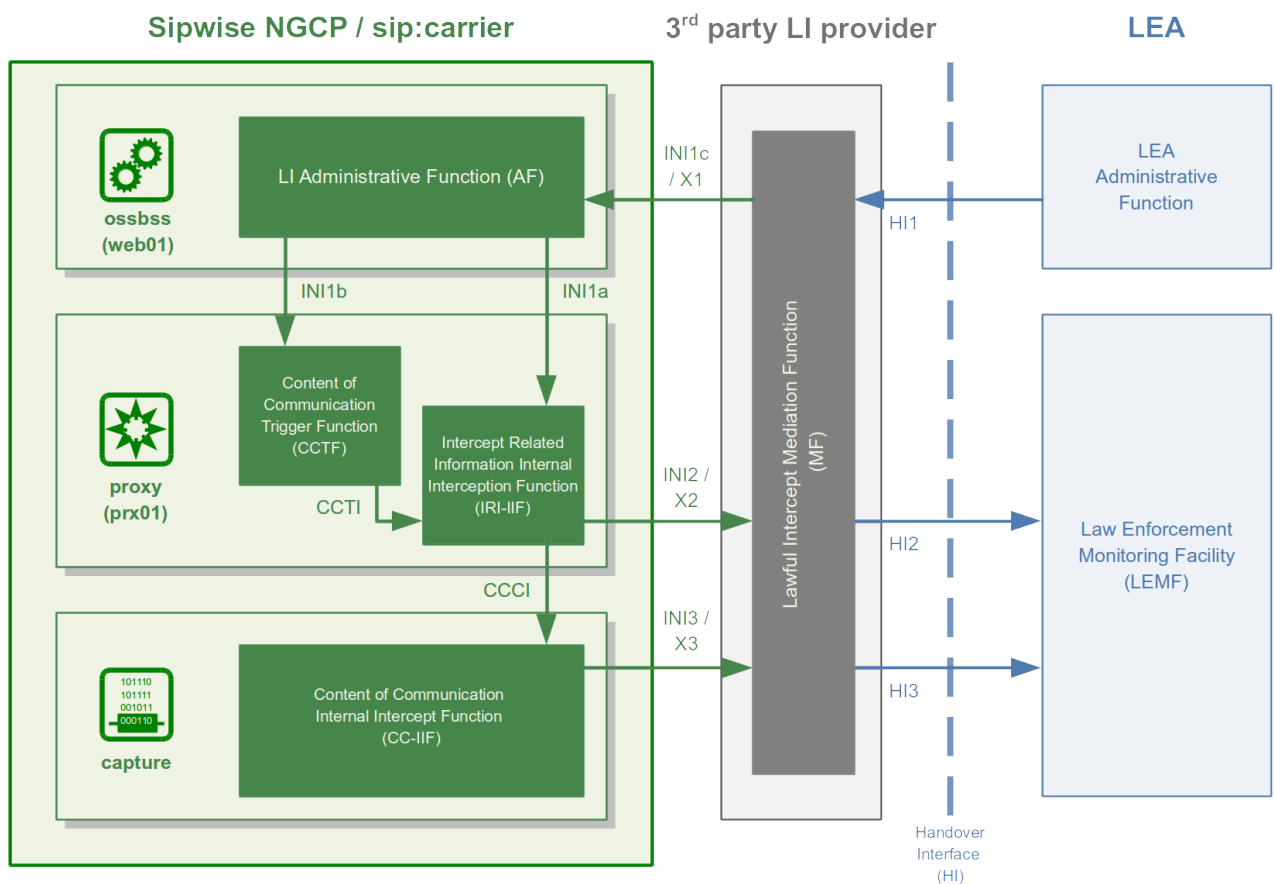


Figure 148: LI with 3rd Party Provider

### 16.3.2.1 Architecture Based on Captagent Module

---

**Note**

This kind of LI implementation will be phased out in future NGCP releases. A short description is kept here for reference, as NGCP still (as of version mr4.5.2) supports LI services with `captagent` module.

---

The `captagent` based implementation of LI functions on Sipwise NGCP includes the following components:

- **captagent**: a software module provided by a third party; its operation can be summarized as follows:
  1. the `captagent` process gets LI requests through an API
  2. the process listens for and analyses SIP (*INVITE*) messages; based on the message headers it decides whether the SIP session must be intercepted
  3. in case the session must be intercepted, `captagent` sends IRI through X2 interface to the MF element
  4. based on the SDP data, the process captures session media and forwards that through X3 interface to the MF element
- **third party MF**: Group2000's LIMA system plays the role of Mediation Function (MF) and interacts with `captagent` module, using X1, X2 and X3 interfaces.

### 16.3.2.2 Architecture Based on Voisniff-NG Module

Although the implementation of LI services with `captagent` is still available and configurable on sip:carrier, Sipwise suggests deploying a revised solution with its `voisniff-ng` software module. This newer implementation also relies on a 3rd party LI provider representing the LI Mediation Function (MF), where Sipwise currently (as of NGCP version mr4.5.2) cooperates with Group2000, Pine and Utimaco.

**Sipwise NGCP components** providing LI functions:

- **ngcp-panel**: this module is responsible for managing REST API for the whole NGCP in general
  - *runs on*: `web01` node on a sip:carrier platform
  - *LI functions*: AF; INI1 / X1 interface towards the MF
- **kamailio-proxy**: this module serves as a generic call control function on the NGCP
  - *runs on*: typically `prx01` node on a sip:carrier platform
  - *LI functions*: CCTF and IRI-IIF; INI2 / X2 interface towards the MF
- **voisniff-ng**: this module is a generic element for capturing SIP and RTP traffic on the NGCP
  - *runs on*: typically `lb01` node on a sip:carrier platform
  - *LI functions*: CC-IIF; INI3 / X3 interface towards the MF

---

**Note**

Please keep in mind that `voisniff-ng` module is not installed by default on Sipwise sip:carrier. Please contact Sipwise if you need to activate LI services on the platform.

---

**Authentication and Confidentiality**

It is required that the communication between the telecommunication operator's network element (that is: Sipwise NGCP) and the MF be authenticated and confidential, since the intercepted session related data and content of communication must not be disclosed to any 3rd party. For this purpose NGCP's LI service applies authentication and LI session data encryption based on public key cryptography mechanism (TLS).

Both Sipwise NGCP and the MF must authenticate themselves by certificates, for this reason the NGCP operator must ensure that valid certificates are deployed on the system. There is a need to contact the 3rd party LI provider, so that he can provide the necessary client certificates that NGCP will use to setup secured connection to the MF on X2 and X3 interfaces.

Similarly, the MF provider must contact the NGCP operator to offer him valid client certificates that the MF element will use to establish secured connection to the NGCP on X1 interface.

**16.3.2.3 Configuration of LI Service**

In order to enable LI services on sip:carrier the platform administrator has to explicitly enable lawful interception through the main configuration file (`config.yml`).

Here below is a sample configuration, which shows parameters of `intercept` and `voisniff` sections.

```
intercept:
  captagent:
    cin_max: '3000'
    cin_min: '0'
    country_code: '49'
    debug: '7'
    filter: 'port 5080'
    license: ''
    port: '18090'
    prefix_len: '3'
    schema: http
  enabled: yes
  peer:
    acc: no
    inbound_prefix: LI_
    outbound_prefix: intercept_
  type: voisniff

voisniff:
  admin_panel: no
  daemon:
    bpf: 'udp or ip6 proto 44 or ip[6:2] & 0x1fff != 0'
```



```

external_interfaces: vlan31 vlan35 vlan61 vlan51
filter:
  exclude:
    - active: '0'
      case_insensitive: '1'
      pattern: '\ncseq: *\d+ +(register|notify|options)'
  include: []
internal_interfaces: lo
li_x1x2x3:
  call_id:
    suffix:
      - _pbx-1
      - _b2b-1
      - _xfer-1
  client_certificate: /etc/ngcp-config/ssl/li/x23_client/x23_client_cert.pem
  enabled: yes
  fix_checksums: no
  fragmented: no
  interface:
    excludes: []
  local_name: sipwise
  private_key: /etc/ngcp-config/ssl/li/x23_client/x23_client_cert_priv_key.pem
  x1:
    port: '18090'
mysql_dump:
  enabled: no
  num_threads: '4'
mysql_dump_threads: '4'
start: yes
threads_per_interface: '10'
partitions:
  increment: '700000'
  keep: '10'

```

### Configuration Parameters

#### **intercept.enable**

Set it to `yes` if you want to activate LI service. Default: `no`

#### **intercept.peer.acc**

Calls to be intercepted may be forwarded to LI peers. The LI peer may forward the call to the original destination, without looping the call back to NGCP. Set this parameter to `yes` if you want to enable billing for such calls. Default: `no`

#### **intercept.peer.inbound\_prefix**

Calls to be intercepted may be forwarded to LI peers. This parameter specifies the prefix that is prepended to SIP usernames when the call is looped back to NGCP, in order to avoid sending the call again to any LI peer. Used by NGCP internally. Default: `LI_`

**intercept.peer.outbound\_prefix**

Calls to be intercepted may be forwarded to LI peers. This parameter specifies the prefix that is prepended to SIP usernames when the call is routed to an LI peer. It will be stripped off by rewrite rules of the peer, before sending the call effectively to the peer. Used by NGCP internally. Default: `intercept_`

**intercept.type**

The LI service provider module; allowed values are:

- `none`: LI service is not activated
- `peer`: LI service is activated and call data capturing is performed by SIP peers
- `captagent`: LI service is activated and call data capturing is performed by `captagent` module
- `voisniff`: LI service is activated and call data capturing is performed by `voisniff` module

Default: `none`

**voisniff.admin\_panel, voisniff.daemon.mysql\_dump.\*, voisniff.partitions.\***

These parameters are not used in LI configuration, but only for call statistics which can be retrieved through the Admin web interface.

**voisniff.daemon.bpf**

This sets the basic packet filter applied by `voisniff-ng` module when capturing packets on network interfaces. Default: `"port 5060 or 5062 or ip6 proto 44 or ip[6:2] & 0x1fff != 0"`

---

**Note**

The default value basically allows capturing SIP traffic only. It is usually necessary to modify the parameter in order to capture both SIP and RTP traffic. An example of such a value: `"udp or ip6 proto 44 or ip[6:2] & 0x1fff != 0"`.

---

**voisniff.daemon.external\_interfaces**

This is a list of network interfaces (typically VLAN IDs) where `voisniff-ng` should listen for and capture packets.

---

**Tip**

VLAN interfaces have to be listed when they are used for intercepted calls. On the other hand virtual interfaces for additional IP addresses (e.g. `eth0:1`) do not have to be listed separately, because the base interface (e.g. `eth0`) will be used to capture packets.

---

**voisniff.daemon.filter.exclude**

Additional filter to determine packets that need to be excluded from capturing. This configuration parameter is a list of items, each of them has 3 components:

- `active`: Determines whether the filter is active or not. Allowed values are: `0` (false/inactive; this is the default) or `1` (true/active).
- `case_insensitive`: Determines whether the `pattern` is case-insensitive (`1`; this is the default) or not (`0`).
- `pattern`: A regular expression providing the matching pattern for packets that have to be filtered.

**voisniff.daemon.filter.include**

Additional filter to determine packets that need to be included in capturing. The parameter has the same syntax as `voisniff.daemon.filter.exclude`.

**voisniff.daemon.internal\_interfaces**

A list of network interfaces which are considered only for internal communication between `voisniff-ng` and other NGCP components. Packets on these interfaces are not captured.

**voisniff.daemon.li\_x1x2x3.call\_id.suffix**

List of NGCP-internal Call-ID suffix patterns that should be ignored when determining the original SIP Call-ID of an intercepted call.

**Caution**

Please do not change these patterns unless instructed to do so by a Sipwise engineer! Changing the patterns may result in falsely recognised Call-IDs and eventually missed SIP messages during an intercepted call.

---

**voisniff.daemon.li\_x1x2x3.client\_certificate**

The client certificate that NGCP uses to connect over TLS to a 3rd party LI provider.

**voisniff.daemon.li\_x1x2x3.enabled**

Set it to `yes` to enable LI services via X1, X2 and X3 interfaces. Default: `no`

**voisniff.daemon.li\_x1x2x3.fix\_checksums**

When enabled (= `yes`), NGCP will calculate UDP header checksum for packets sent out on X2 and X3 interfaces. This is necessary when the checksum calculation is normally left to the network interface hardware and therefore the UDP header checksum is inherently incorrect on application level. Also the UDP checksum must be calculated by `voisniff-ng` on re-assembled packets, so enable this option if there are fragmented packets in intercepted call traffic. Default: disabled (= `no`)

**voisniff.daemon.li\_x1x2x3.fragmented**

When disabled (= `no`), `voisniff-ng` defragments all packets and sends out only reassembled packets via X2 and X3 interfaces. If the option is enabled (= `yes`), `voisniff-ng` will instead send out the original fragments via X2 and X3. Default: `no`

**voisniff.daemon.li\_x1x2x3.interface.excludes**

This is a list of interfaces that must be excluded from the interception procedures. The list contains regular expressions that describe the to-be-excluded interfaces, for example: `^lo$` to exclude the loopback interface. Default: empty list

**voisniff.daemon.li\_x1x2x3.local\_name**

This parameter maps to the `header.source` field of the X2 protocol. It's an arbitrary string and can be used to identify the sending NGCP system. Default: `sipwise`

**Note**

As of NGCP version mr4.5.2, this is currently not used.

---

**voisniff.daemon.li\_x1x2x3.private\_key**

The private key that NGCP uses to connect over TLS to a 3rd party LI provider. Only necessary if the client certificate file does not include the private key.

**voisniff.daemon.li\_x1x2x3.x1.port**

The port number on which `voisniff-ng` listens for incoming X1 messages. Default: 18090

**Caution**

You should leave the parameter set to the default value, unless there is a good reason to change it. The default value ensures backward compatibility with `captagent` LI module.

**voisniff.daemon.start**

Determines whether `voisniff` service must be started on the platform. Set it to `yes` if you'd like to activate `voisniff` that is needed for LI service too. Default: `no`

**voisniff.daemon.threads\_per\_interface**

This is a performance tuning option and controls how many threads per enabled sniffing interface should be launched. Example: if it's set to 10 and 3 interfaces are enabled for sniffing, a total of 30 threads will be launched. Default: 2

**Caution**

Do not set it to a high number, or simply leave it at its default value, unless there is a performance problem with `voisniff` service. Please keep in mind that a high number of threads might also decrease the overall system performance of NGCP!

**16.3.3 X1, X2 and X3 Interface Specification**

Short description of  $X_n$  interfaces:

- The **X1** interface is used by an LI provider to create, modify, delete and list interceptions on the Sipwise NGCP. It is designed as RESTful HTTP interface using JSON (with JSON-HAL in responses from the NGCP) as content type to provision interceptions.
- The **X2** interface is a TLV based interface with JSON payload with a simple request/response mechanism over a secure TLS connection, used to pass intercepted signaling data towards an LI provider.
- The **X3** interface is also a TLV based interface with a binary payload encapsulating the intercepted RTP data.

**16.3.3.1 X1 Interface**

The resource used to work with interceptions is always `https://ngcp-ip:1443/api/interceptions/`

**Authentication**

Authentication and authorization on the NGCP API is performed via HTTP Basic Auth or SSL Client certificates.

- **HTTP Basic Auth:** With `cURL` use `--user username:password` option to specify your access credentials.

```
curl -i -X GET --user myuser:mypassword https://example.org:1443/api/interceptions/
```

Additionally use the `--insecure` option if you are testing against a self-signed server certificate.

- **SSL Client Authentication:** You can generate and download client certificates for administrators and resellers via the NGCP Panel in the Administrators view.

For the actual client authentication, you will need two files which you can download from the panel after creating the client certificates:

1. The client certificate generated via the NGCP Panel. This is usually labelled NGCP-API-client-certificate-xxxxx.pem.
2. The CA certificate used to sign the server certificate, in case it as been self-signed or the CA is not recognized by the client host environment.

With *cURL* use `--cert /path/to/NGCPAPIclientcertificatexxxxx.pem` to specify the client certificate, and `-cacert /path/to/cacert.pem` to specify the CA certificate in case of a self-signed server certificate.

```
curl -i -X GET --cert /path/to/NGCPAPIclientcertificatexxxxx.pem \
--cacert /path/to/cacert.pem https://example.org:1443/api/interceptions/
```

Additionally use the `--insecure` option if you are testing against a self-signed server certificate.

## API Description

### Collection Actions

Allowed methods for the collection as in `METHOD /api/interceptions/`

- OPTIONS
- POST
- GET
- HEAD

### Item Actions

Allowed methods for a collection item as in `METHOD /api/interceptions/id`

- PATCH
- OPTIONS
- DELETE
- PUT
- GET
- HEAD

### Properties

- `liid` (Number): The LI ID for this interception.
- `number` (String): The number to intercept.
- `x2_host` (String): The IP address of the X2 interface.
- `x2_password` (null, String): The password for authenticating on the X2 interface.
- `x2_port` (Number): The port of the X2 interface.
- `x2_user` (null, String): The username for authenticating on the X2 interface.

- `x3_host` (null, String): The IP address of the X3 interface.
- `x3_port` (null, Number): The port of the X3 interface.
- `x3_required` (null, Boolean): Whether to also intercept call content via X3 interface (*false* by default).

### Query Parameters

- `liid`: Filter for interceptions of a specific interception ID
- `number`: Filter for interceptions of a specific number (in E.164 format)
- `order_by`: Order collection by a specific attribute. Possible values are: `id`, `reseller_id`, `liid`, `number`, `cc_required`, `delivery_host`, `delivery_port`, `delivery_user`, `delivery_pass`, `modify_timestamp`, `create_timestamp`, `deleted`, `uuid`, `sip_username`, `sip_domain`, `cc_delivery_host`, `cc_delivery_port`
- `order_by_direction`: Direction which the collection should be ordered by. Possible values are: `asc` (default), `desc`

### API Examples

#### Get a specific interception

- Request:

```
curl -i --insecure --user administrator:administrator -X GET
https://localhost:1443/api/interceptions/528
```

- Response:

```
HTTP/1.1 200 OK
Server: nginx
Date: Tue, 01 Dec 2015 09:43:41 GMT
ContentType: application/hal+json; profile="http://purl.org/sipwise/ngcpapi/";
  charset=utf8
ContentLength: 634
Connection: keepalive
Link: </api/interceptions/>; rel=collection
Link: <http://purl.org/sipwise/ngcpapi/>; rel=profile
Link: </api/interceptions/528>; rel="item self"
SetCookie: ngcp_panel_session=35b56d921c36c1fc6edb8fcd0a86dd9af61ec62a; path=/;
  expires=Tue, 01 Dec 2015 10:43:41 GMT; HttpOnly
StrictTransportSecurity: maxage=15768000
{
  "_links" : {
    "collection" : {
      "href" : "/api/interceptions/"
    },
  },
  "curies" : {
    "href" : "http://purl.org/sipwise/ngcpapi/#rel{rel}",
    "name" : "ngcp",
```

```

    "templated" : true
  },
  "profile" : {
    "href" : "http://purl.org/sipwise/ngcpapi/"
  },
  "self" : {
    "href" : "/api/interceptions/528"
  }
},
"id" : 528,
"liid" : 918273,
"number" : "0014155550132",
"x2_host" : "192.168.42.42",
"x2_password" : null,
"x2_port" : 3002,
"x2_user" : null,
"x3_host" : "192.168.42.42",
"x3_port" : 3003,
"x3_required" : true
}

```

### Get all interceptions for a number

- Request:

```

curl -i --insecure --user administrator:administrator -X GET \
https://localhost:1443/api/interceptions/?number=0014155550132

```

- Response:

```

HTTP/1.1 200 OK
Server: nginx
Date: Tue, 01 Dec 2015 09:47:36 GMT
ContentType: application/hal+json; profile="http://purl.org/sipwise/ngcpapi/";
  charset=utf8
ContentLength: 1283
Connection: keepalive
SetCookie: ngcp_panel_session=238550c5737058db619b183d925b5f9a61261cfe; path=/;
  expires=Tue, 01 Dec 2015 10:47:36 GMT; HttpOnly
StrictTransportSecurity: maxage=15768000
{
  "_embedded" : {
    "ngcp:interceptions" : {
      "_links" : {
        "collection" : {
          "href" : "/api/interceptions/"
        },
        "curies" : {

```

```

        "href" : "http://purl.org/sipwise/ngcpapi/#rel{rel}",
        "name" : "ngcp",
        "templated" : true
    },
    "profile" : {
        "href" : "http://purl.org/sipwise/ngcpapi/"
    },
    "self" : {
        "href" : "/api/interceptions/520"
    }
},
"id" : 520,
"liid" : 1,
"number" : "0014155550132",
"x2_host" : "192.168.42.42",
"x2_password" : null,
"x2_port" : 3002,
"x2_user" : null,
"x3_host" : "192.168.42.42",
"x3_port" : 3003,
"x3_required" : true
}
},
"_links" : {
    "curies" : {
        "href" : "http://purl.org/sipwise/ngcpapi/#rel{rel}",
        "name" : "ngcp",
        "templated" : true
    },
    "ngcp:interceptions" : {
        "href" : "/api/interceptions/520"
    },
    "profile" : {
        "href" : "http://purl.org/sipwise/ngcpapi/"
    },
    "self" : {
        "href" : "/api/interceptions/?page=1&rows=10"
    }
},
"total_count" : 1
}

```

### Get all interceptions for all numbers

- Request:

```
curl -i --insecure --user administrator:administrator -X GET \
```



```
https://localhost:1443/api/interceptions/
```

- **Response:**

```
HTTP/1.1 200 OK
Server: nginx
Date: Tue, 01 Dec 2015 09:43:18 GMT
ContentType: application/hal+json; profile="http://purl.org/sipwise/ngcpapi/";
  charset=utf8
ContentLength: 2364
Connection: keepalive
SetCookie: ngcp_panel_session=68398eea5bdd3885ad0517e1f6d367ccc80111fa; path=/;
  expires=Tue, 01 Dec 2015 10:43:18 GMT; HttpOnly
StrictTransportSecurity: maxage=15768000
{
  "_embedded" : {
    "ngcp:interceptions" : [
      {
        "_links" : {
          "collection" : {
            "href" : "/api/interceptions/"
          },
          "curies" : {
            "href" : "http://purl.org/sipwise/ngcpapi/#rel{rel}",
            "name" : "ngcp",
            "templated" : true
          },
          "profile" : {
            "href" : "http://purl.org/sipwise/ngcpapi/"
          },
          "self" : {
            "href" : "/api/interceptions/520"
          }
        },
        "id" : 520,
        "liid" : 1,
        "number" : "0014155550132",
        "x2_host" : "192.168.42.42",
        "x2_password" : null,
        "x2_port" : 3002,
        "x2_user" : null,
        "x3_host" : "192.168.42.42",
        "x3_port" : 3003,
        "x3_required" : true
      },
      {
        "_links" : {
          "collection" : {
```

```
        "href" : "/api/interceptions/"
      },
      "curies" : {
        "href" : "http://purl.org/sipwise/ngcpapi/#rel{rel}",
        "name" : "ngcp",
        "templated" : true
      },
      "profile" : {
        "href" : "http://purl.org/sipwise/ngcpapi/"
      },
      "self" : {
        "href" : "/api/interceptions/528"
      }
    },
    "id" : 528,
    "liid" : 918273,
    "number" : "0014155550132",
    "x2_host" : "192.168.42.42",
    "x2_password" : null,
    "x2_port" : 3002,
    "x2_user" : null,
    "x3_host" : "192.168.42.42",
    "x3_port" : 3003,
    "x3_required" : true
  }
]
},
"_links" : {
  "curies" : {
    "href" : "http://purl.org/sipwise/ngcpapi/#rel{rel}",
    "name" : "ngcp",
    "templated" : true
  },
  "ngcp:interceptions" : [
    {
      "href" : "/api/interceptions/520"
    },
    {
      "href" : "/api/interceptions/528"
    }
  ],
  "profile" : {
    "href" : "http://purl.org/sipwise/ngcpapi/"
  },
  "self" : {
    "href" : "/api/interceptions/?page=1&rows=10"
  }
},
"total_count" : 2
```

```
}
```

### Get interception for specific LIID

- Request:

```
curl -i --insecure --user administrator:administrator -X GET \  
https://localhost:1443/api/interceptions/?liid=9876
```

- Response:

```
HTTP/1.1 200 OK  
Server: nginx  
Date: Tue, 01 Dec 2015 09:50:41 GMT  
ContentType: application/hal+json; profile="http://purl.org/sipwise/ngcpapi/";  
  charset=utf8  
ContentLength: 1283  
Connection: keepalive  
SetCookie: ngcp_panel_session=23960dde6bb90f0c5c84575890194c53cce120ce; path=/  
  expires=Tue, 01 Dec 2015 10:50:40 GMT; HttpOnly  
StrictTransportSecurity: maxage=15768000  
{  
  "_embedded" : {  
    "ngcp:interceptions" : {  
      "_links" : {  
        "collection" : {  
          "href" : "/api/interceptions/"  
        },  
        "curies" : {  
          "href" : "http://purl.org/sipwise/ngcpapi/#rel{rel}",  
          "name" : "ngcp",  
          "templated" : true  
        },  
        "profile" : {  
          "href" : "http://purl.org/sipwise/ngcpapi/"  
        },  
        "self" : {  
          "href" : "/api/interceptions/520"  
        }  
      },  
      "id" : 520,  
      "liid" : 1,  
      "number" : "0014155550132",  
      "x2_host" : "192.168.42.42",  
      "x2_password" : null,  
      "x2_port" : 3002,  
      "x2_user" : null,  
      "x3_host" : "192.168.42.42",
```

```

    "x3_port" : 3003,
    "x3_required" : true
  }
},
"_links" : {
  "curies" : {
    "href" : "http://purl.org/sipwise/ngcpapi/#rel{rel}",
    "name" : "ngcp",
    "templated" : true
  },
  "ngcp:interceptions" : {
    "href" : "/api/interceptions/520"
  },
  "profile" : {
    "href" : "http://purl.org/sipwise/ngcpapi/"
  },
  "self" : {
    "href" : "/api/interceptions/?page=1&rows=10"
  }
},
"total_count" : 1
}

```

### Create interception for a specific number

- Request:

```

curl -i --insecure --user administrator:administrator -X POST \
-H "ContentType: application/json" --data \
'{"liid":123, "number":"31032222203", "x2_host":"127.0.0.1", "x2_port":12345,
  "x3_required":true, "x3_host":"127.0.0.2", "x3_port":23456}' \
https://localhost:1443/api/interceptions/

```

- Response:

```

HTTP/1.1 201 Created
TransferEncoding: chunked
Connection: close
Location: /api/interceptions/528
SetCookie: ngcp_panel_session=e7817079d121fae4d86448b10e1fa21d0201c526; path=/;
  expires=Tue, 01 Dec 2015 10:43:18 GMT; HttpOnly
StrictTransportSecurity: maxage=15768000

```

The path to the newly created interception is found in the *Location* header of the response.

### Update specific interception

- Request:

```
curl -i --insecure --user administrator:administrator -X PUT \
-H "ContentType: application/json" -H 'Prefer: return=representation' --data \
'{"liid":918273, "number":"0014155550132", "x2_host":"192.168.42.42", "x2_port":5000,
  "x3_required":false}' \
https://localhost:1443/api/interceptions/123
```

- **Response:**

```
HTTP/1.1 200 OK
ContentType: application/hal+json; profile="http://purl.org/sipwise/ngcpapi/";
  charset=utf8
ContentLength: 621
Link: </api/interceptions/>; rel=collection
Link: <http://purl.org/sipwise/ngcpapi/>; rel=profile
Link: </api/interceptions/530>; rel=self
PreferenceApplied: return=representation
SetCookie: ngcp_panel_session=0b56e4a197b0e9f6e22a998e85473a0184770740; path=/;
  expires=Tue, 01 Dec 2015 10:56:17 GMT; HttpOnly
{
  "_links" : {
    "collection" : {
      "href" : "/api/interceptions/"
    },
    "curies" : {
      "href" : "http://purl.org/sipwise/ngcpapi/#rel{rel}",
      "name" : "ngcp",
      "templated" : true
    },
    "profile" : {
      "href" : "http://purl.org/sipwise/ngcpapi/"
    },
    "self" : {
      "href" : "/api/interceptions/530"
    }
  },
  "id" : 530,
  "liid" : 918273,
  "number" : "0014155550132",
  "x2_host" : "192.168.42.42",
  "x2_password" : null,
  "x2_port" : 5000,
  "x2_user" : null,
  "x3_host" : null,
  "x3_port" : null,
  "x3_required" : false
}
```

The *Prefer: return=representation* header forces the API to return the content, otherwise status 201 with no content is returned.

### Update only certain items for a specific interception

- Request:

```
curl -i --insecure --user administrator:administrator -X PATCH \  
-H "ContentType: application/jsonpatch+json" -H 'Prefer: return=representation' \  
--data '[{"op": "replace", "path": "/x2_host", "value": "192.168.42.42"}, {"op": "replace", \  
"path": "/x2_port", "value": 4000}]' \  
https://localhost:1443/api/interceptions/530
```

- Response:

```
HTTP/1.1 200 OK  
Server: nginx  
Date: Tue, 01 Dec 2015 10:06:06 GMT  
ContentType: application/hal+json; profile="http://purl.org/sipwise/ngcpapi/";  
charset=utf8  
ContentLength: 620  
Connection: close  
Link: </api/interceptions/>; rel=collection  
Link: <http://purl.org/sipwise/ngcpapi/>; rel=profile  
Link: </api/interceptions/530>; rel=self  
PreferenceApplied: return=representation  
SetCookie: ngcp_panel_session=0693129d63d543a85f96d464ff9a8f807cfc4d18; path=/  
expires=Tue, 01 Dec 2015 11:06:06 GMT; HttpOnly  
StrictTransportSecurity: maxage=15768000  
{  
  "_links" : {  
    "collection" : {  
      "href" : "/api/interceptions/"  
    },  
    "curies" : {  
      "href" : "http://purl.org/sipwise/ngcpapi/#rel{rel}",  
      "name" : "ngcp",  
      "templated" : true  
    },  
    "profile" : {  
      "href" : "http://purl.org/sipwise/ngcpapi/"  
    },  
    "self" : {  
      "href" : "/api/interceptions/530"  
    }  
  },  
  "id" : 530,  
  "liid" : 918273,  
  "number" : "0014155550132",  
  "x2_host" : "192.168.42.42",  
  "x2_password" : null,  
  "x2_port" : 4000,
```

```

"x2_user" : null,
"x3_host" : null,
"x3_port" : null,
"x3_required" : false
}

```

### Delete specific interception

- Request:

```

curl -i --insecure --user administrator:administrator -X DELETE \
https://localhost:1443/api/interceptions/123

```

- Response:

```

HTTP/1.1 204 No Content
Server: nginx
Date: Tue, 01 Dec 2015 10:08:49 GMT
Connection: keepalive
Set-Cookie: ngcp_panel_session=570c66b66732629766f86b8ed9bd0d64902ae73e; path=/;
    expires=Tue, 01 Dec 2015 11:08:49 GMT; HttpOnly
XCatalyst: 5.90042
StrictTransportSecurity: maxage=15768000

```

#### 16.3.3.2 X2 Interface

The communication via the X2 interface consists of request-response pairs.

##### **Request**

The request is formatted as: X2/<bodylength>/<body>

**Body** part has the following items:

Table 24: X2 Message Body Items

Element	Type	Length	Description
/x2/header/source	String	arbitrary length	identifier of Sipwise node which captured the data
/x2/header/destination	String	arbitrary length	identifier of LI mediation system
/x2/header/type	String	arbitrary length	always "sip" (but later potentially "xmpp" and others too)
/x2/header/version	PosInteger	arbitrary length	always "1"

Table 24: (continued)

Element	Type	Length	Description
/x2/header/timestamp	String	27 chars	format: YYYY-MM-DDThh:mm:ss.ffffffZ; timestamp in UTC when the X2 package is sent to mediation
/x2/body/dialogid	PosInteger	arbitrary length	globally increasing counter for each new communication dialog (e.g. call)
/x2/body/messageid	PosInteger	arbitrary length	increasing counter for each new x2 message within a dialog, starting from 0
/x2/body/timestamp	String	27 chars	format: YYYY-MM-DDThh:mm:ss.ffffffZ; timestamp in UTC when the package has been captured on the wire
/x2/body/interceptions			one or more elements containing the following information, one element per intercepted target:
/x2/body/interceptions/liid	PosInteger	arbitrary length	interception id ("liid") as set via X1 interface
/x2/body/interceptions/direction	String	arbitrary length	either "totarget" or "fromtarget" from the soft-switch perspective (if target is the called party, it is "totarget", if target is the calling party, it is "fromtarget").
/x2/body/data	Base64 encoded	arbitrary	content of full IP frame and up on the OSI layer; packets fragmented on the wire are provided in fully assembled format

**Example of full message:**

```
X2/418/
{
  "header": {
    "source": "prx01a.example.com",
    "destination": "x2destination.example.com",
    "type": "sip",
    "version": 1,
    "timestamp": "2015 03 11T09:18:04.729803Z"
  },
  "body": {
    "dialogid": 4,
    "messageid": 0,
    "timestamp": "2015 03 11T09:18:04.729123Z",
    "interceptions": [
      { "liid": 174, "direction": "fromtarget" },
      { "liid": 175, "direction": "totarget" }
    ],
    "data": "<base64 encoded ip,udp/tcp,sip frame>"
  }
}
```



**Response**

- Success: X2-ACK/0/
- Error: X2-ERR/<length>/<error string>

**Keep-Alive Mechanism**

A regular keep-alive mechanism with a default value of 10s is used on the connection if it is re-used across multiple messages.

- Request: X2/0/
- Response: X2-ACK/0/

**16.3.3.3 X3 Interface**

On the X3 interface TLV based packets are sent via secured (TLS) connection on a pre-established stream. X3 messages do not need to be acknowledged, except for keep-alive messages.

**X3 Message Structure**

Table 25: X3 Message Structure

Field	Length
Header	arbitrary
CCCID	4 bytes
MessageId	4 bytes
Timestamp	8 bytes
Payload	arbitrary

**Header Details**

Table 26: X3: Header Details

Field	Length	Content
type	2 bytes	always "X3"
delimiter	1 byte	always "/"
length	arbitrary	ASCII string
delimiter	1 byte	always "/"

**CCCID Details**

`dialogid` (32 bit in network byte order, reset to 0 after  $2^{32}-1$ )

The `dialogid` is referencing the `/x2/body/dialogid` field in order to correlate an X3 packet to an X2 call.

**MessageId Details**

`messageid` (32 bit in network byte order, reset to 0 after  $2^{32}-1$ )

The `messageid` is a counter within a dialog sequencing the X3 packets sent from the NGCP. This counter is not correlated in any way with X2, rather than starting at 0 with the first RTP packet captured within a dialog.

**Timestamp Details**

- `seconds` (32 bit in network byte order)
- `fraction` (32 bit in network byte order)

The timestamp represents the Unix epoch starting from 1970-01-01.

**Payload Details**

Table 27: X3: Payload Details

Field	Length
original ip header	20 bytes for v4, 40 bytes for v6
original udp header	8 bytes
original rtp header	variable, 12-72 bytes
original rtp payload	arbitrary

**Keep-Alive Mechanism**

A regular keep-alive mechanism with a default value of 10s is used on the connection if it is re-used across multiple messages.

- Request: `X3/0/`
- Response: `X3-ACK/0/`

**16.4 3rd Party Call Control****16.4.1 Introduction**

The Sipwise NGCP offers the possibility to perform call control through 3rd party applications. This functionality, called **Party Call Control** and referred to as "**PCC**" throughout this handbook, is available since mr5.1.1 release.

**Incoming calls** to local subscribers may be signalled to a 3rd party CAC (Call Admission Control) server. Before accepting (that is: sending the SIP *INVITE* request to the called subscriber) or rejecting the call, NGCP will wait for an explicit reply from the CAC / PCC server, or a timeout.

**Short Messages** received by NGCP for a local subscriber may also be signalled to the PCC server. After an explicit reply with "accepted" status from the PCC server, the NGCP will forward the SM to the final recipient.



**Important**

Sipwise NGCP does not support delivering SMS to the local subscribers directly. Local subscribers can define a *Call Forward for SMS* instead, thus allowing themselves to receive SMS on their mobile phones.

3rd party call control may be implemented in many ways, such as by server-side or client-side applications (e.g. smartphone app).

**Note**

Please note that the Sipwise NGCP implements a proprietary protocol for PCC deployments and adapting the protocol to customer needs requires software development from Sipwise.

**16.4.2 Details of Call Processing with PCC**

**16.4.2.1 Overview**

The following figure presents the schema of incoming call processing when PCC is involved:

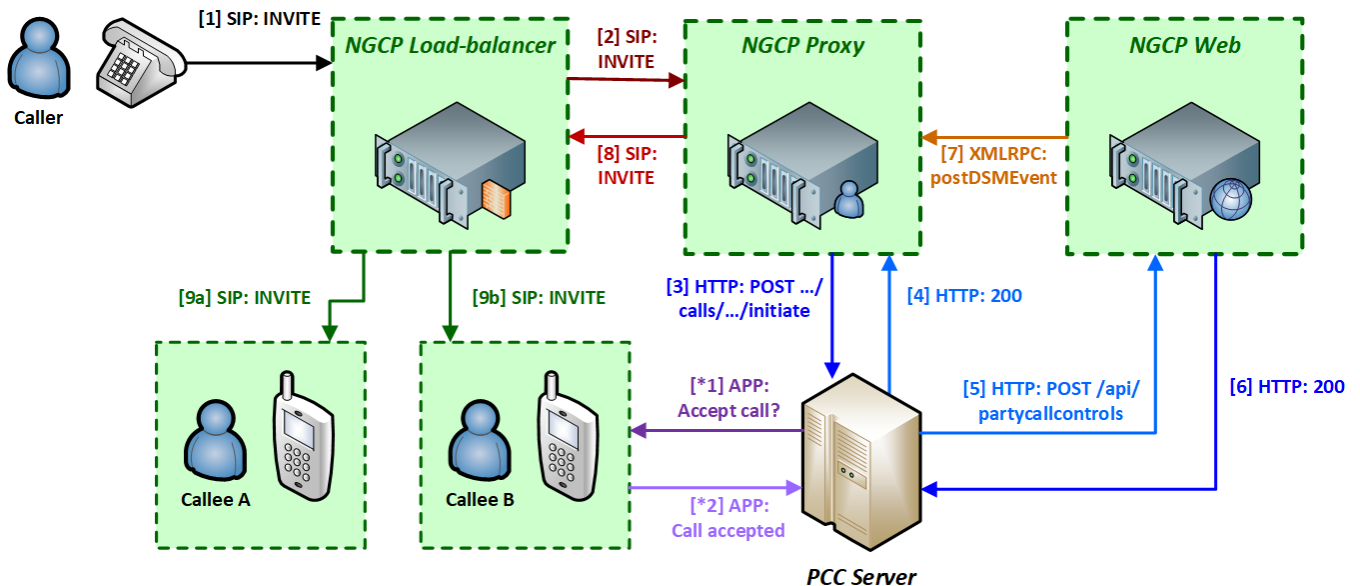


Figure 149: Overview of Party Call Control

The messages / interactions of PCC call processing are:

1. NGCP Load-Balancer receives a SIP *INVITE* message from the caller.
2. The LB forwards the *INVITE* to the PROXY component as usual with every incoming call.
3. The PROXY (*kamailio-proxy* module) checks whether the called subscriber has the PCC feature activated. If this is the case, it will send an HTTP *POST* or *GET* request (configurable) to the PCC server with the most important details of the call (such as calling and called party numbers, call-ID, a token for internal identification of the session).
4. The PCC server replies with *200 OK* HTTP status in order to indicate that it understood the request and will provide the final status (such as *ACCEPTED* or *REJECTED*) of the call later.
  - *Optional:*
    - \*1) The PCC server requests the subscriber's confirmation to accept the call for instance via a smartphone app.
    - \*2) The subscriber indicates accepting the call to the PCC server.
5. The PCC server send an HTTP *POST* request to the WEB component of NGCP, using the NGCP REST API, to signal accepting the call.
6. The WEB will reply with *200 OK* HTTP status.
7. The WEB sends an internal XMLRPC request to PROXY indicating that the incoming call can be accepted.
8. The PROXY sends the SIP *INVITE* message to the LB, i.e. it continues the call setup as usual.
9. The LB sends the *INVITE* to the subscriber.

There are more software modules within NGCP's components and those are shown separately on the diagrams in following sections of the handbook. For instance the PROXY component has the *kamailio-proxy* and *ngcp-sems* modules.

#### 16.4.2.2 Successful Call Initiation at PCC Server

A subscriber with PCC activated will not receive the SIP *INVITE* request directly, but only after a series of intermediate CAC (Call Admission Control) steps, involving the NGCP Proxy and the PCC server. First of those steps is the call initiation at the PCC server:

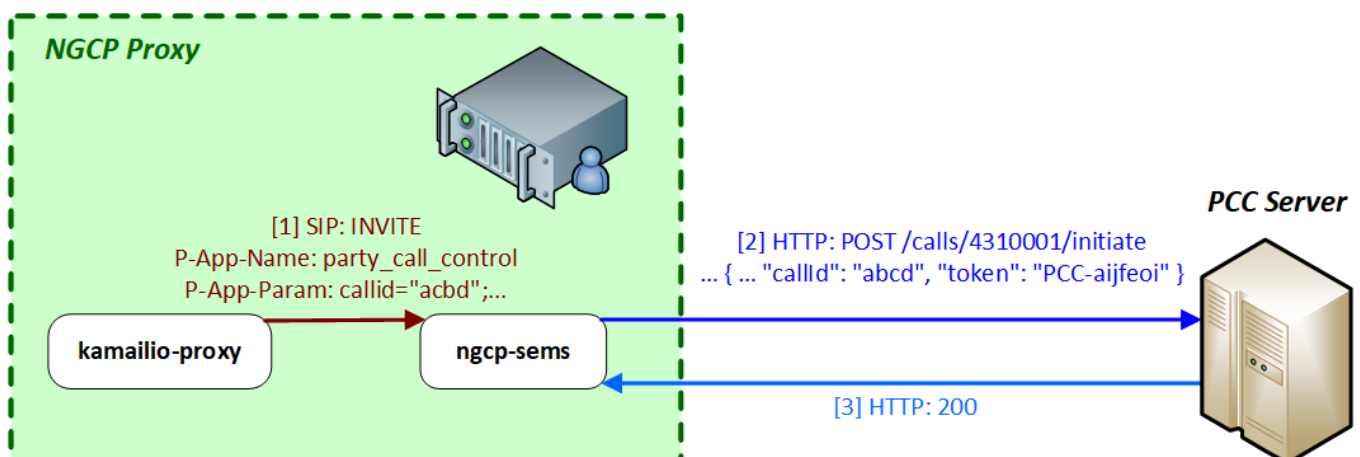


Figure 150: Successful Call Initiation with PCC

1. When *kamailio-proxy* receives the *INVITE* request from NGCP LB, it will forward the message to *ngcp-sems* module with 2 private SIP headers:

```
P-App-Name: party_call_control
P-App-Param: callid="abcd";caller="4369912345";callee="4310001";caller_clir="0";
```

2. These headers will activate the PCC function in *ngcp-sems* and it will send an HTTP *POST* request to the PCC server, instead of creating the second call leg directly towards NGCP LB. An example of such a request (not all details included):

```
POST /calls/4310001/initiate HTTP/1.1
Content-Type: application/json

{
  "actualMsisdn": 4369912345,
  "callingMsisdn": 4310001,
  "actualClir": 0,
  "callId": "abcd",
  "token": "PCC-aijfeoi"
}
```

where:

- `actualMsisdn`: calling party number
- `callingMsisdn`: called party number
- `actualClir`: non-0 if CLIR is active
- `callid`: the SIP Call-ID
- `token`: a generated token that identifies the session between NGCP and the PCC server

The target URL has the format: `/calls/<called_party_num>/initiate`

3. The PCC server replies with HTTP *200 OK* if it understood the request and can proceed with working on that.

#### 16.4.2.3 Call Initiation at PCC Server with Error

The *ngcp-sems* module on NGCP Proxy will wait for a response from PCC server, once it has sent the "initiate" request to it. If the PCC server responds with an HTTP error status, such as any *4xx*, then *ngcp-sems* reports the error condition of PCC server with a SIP *487 Request Terminated* reply to *kamailio-proxy*.

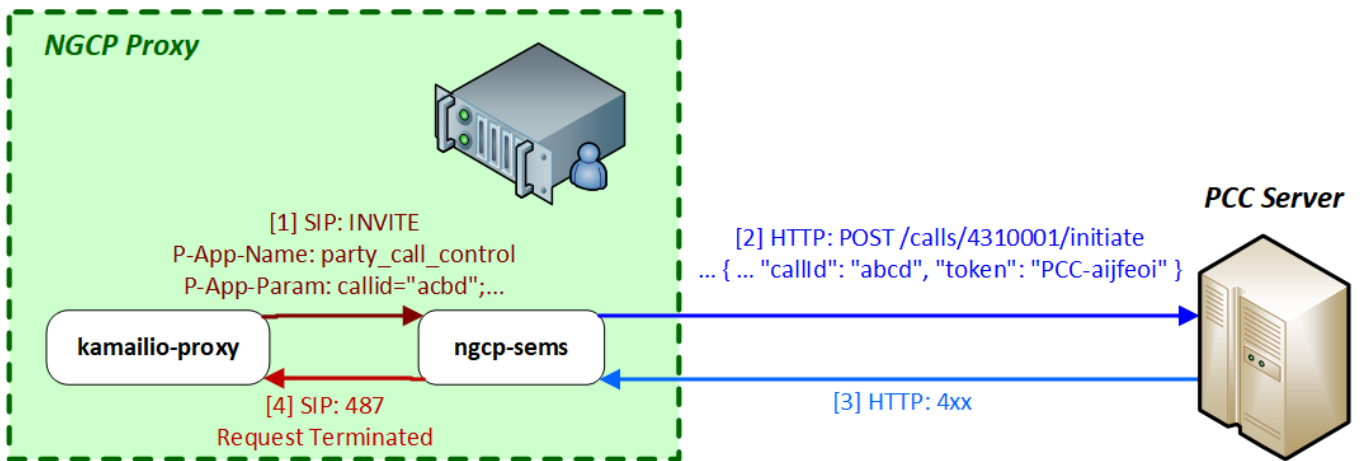


Figure 151: Call Initiation Error with PCC

#### 16.4.2.4 Call Initiation at PCC Server with Timeout

The *ngcp-sems* module on NGCP Proxy will wait for a response from PCC server, once it has sent the "initiate" request to it. If the PCC server does not respond with *HTTP 200 OK* within 30 seconds (configurable) then *ngcp-sems* considers the PCC is not available. In such a case *ngcp-sems* sends a *SIP 408 Timeout* reply to *kamailio-proxy*.

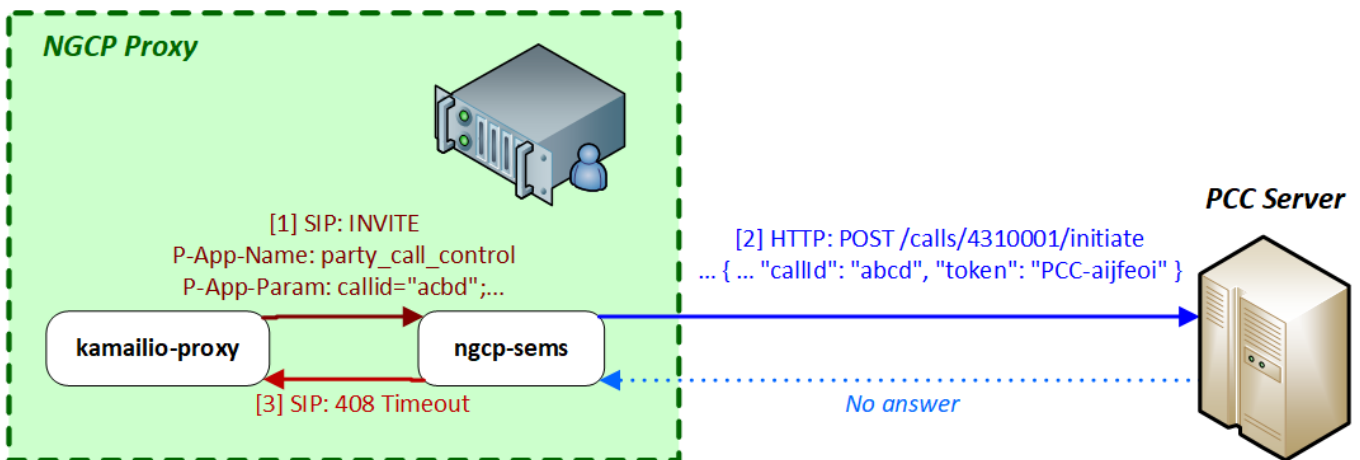


Figure 152: Call Initiation Timeout with PCC

#### 16.4.2.5 Call Accepted by PCC Server

If the PCC server (eventually this may also be the called subscriber) accepts the call, the PCC server will send an *HTTP POST* request to the REST API interface of NGCP (Web/Management component). This request must contain a `status` field with the content `ACCEPT` (configurable) so that NGCP continues the call setup towards called party. Example:

```
POST /api/partycallcontrols HTTP/1.1
Content-Type: application/json
```

```
{
  "type": "pcc",
  "caller": 4369912345,
  "callee": 4310001,
  "status": "ACCEPT",
  "callId": "abcd",
  "token": "PCC-aijfeoi"
}
```

The target URL of the request: `/api/partycallcontrols`. The `type` parameter must have a value of `pcc`.

You can see the flow of messages in the diagram below:

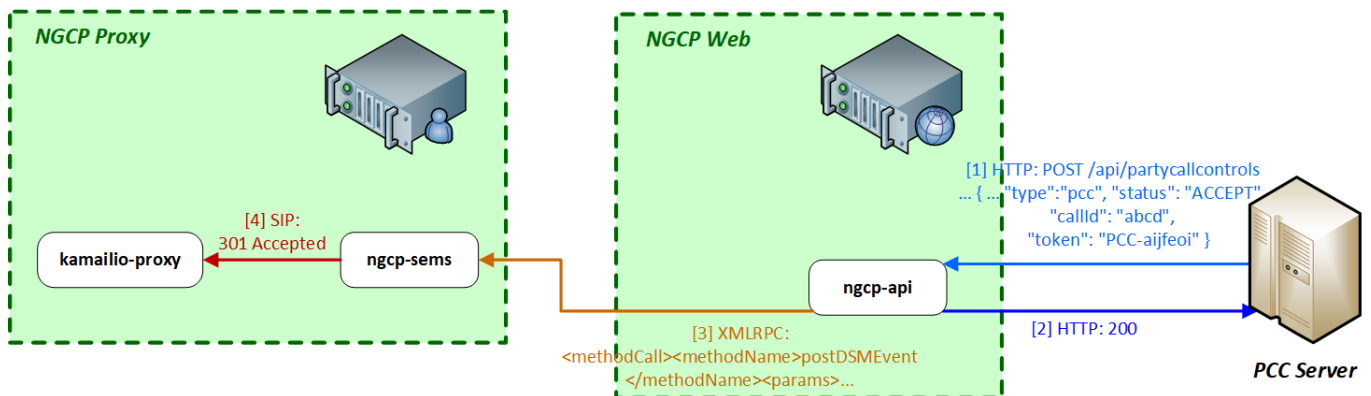


Figure 153: Call Accepted by PCC

1. The PCC server sends an HTTP *POST* request to NGCP's REST API.
2. NGCP Web will reply with *200 OK* HTTP status once the request is validated.
3. The *ngcp-panel* module generates an XMLRPC call to the *ngcp-sems* module on the PROXY. An example is shown here:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>postDSMEEvent</methodName>
  <params>
    <param>
      <value><string>PCC-aijfeoi</string></value>
    </param>
    <param>
      <value><array><data>
        <value><array><data>
          <value><string>cmd</string></value>
          <value><string>handleCall</string></value>
        </data></array></value>
        <value><array><data>
          <value><string>callid</string></value>
          <value><string>abcd</string></value>
        </data></array></value>
      </value>
    </param>
  </params>
</methodCall>
```

```
</data></array></value>
<value><array><data>
  <value><string>caller</string></value>
  <value><string>4369912345</string></value>
</data></array></value>
<value><array><data>
  <value><string>callee</string></value>
  <value><string>4310001</string></value>
</data></array></value>
<value><array><data>
  <value><string>status</string></value>
  <value><string>ACCEPT</string></value>
</data></array></value>
</data></array></value>
</param>
</params>
</methodCall>
```

At this point *ngcp-sems* examines the following:

- whether the token (listed as first `param` parameter of `postDSMEvent`) matches any of the saved session tokens
  - whether the `callid` parameter's value matches the session's SIP Call-ID
  - whether the `status` parameter's value is `ACCEPT` (configurable)  
and if all those conditions are valid it will indicate to *kamailio-proxy* module that the call can be accepted (i.e. call setup towards the callee may continue).
4. *ngcp-sems* module sends *301 Accepted* SIP response to *kamailio-proxy* and the latter can forward the SIP *INVITE* message to NGCP LB. If the `status` parameter's value is not `ACCEPT` (configurable), *ngcp-sems* will reply *487 Request Terminated* to *kamailio-proxy*.

#### 16.4.2.6 Indicating Call Termination at PCC Server

In the same manner as call initiation happens, call termination is also reported by NGCP towards the PCC server.



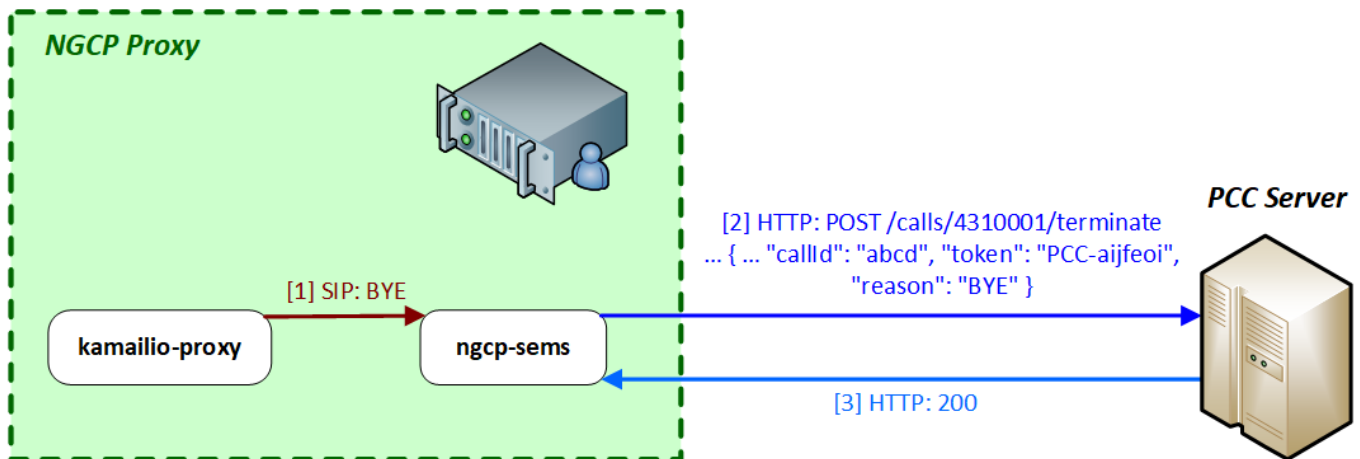


Figure 154: Call Termination with PCC

The target URL of the HTTP *POST* request for the call termination case looks like: `/calls/<called_party_num>/terminate`

The body of the request must contain the following element: `"reason": "BYE"`, where the reason can be one of `BYE`, `CANCEL`, `NOANSWER` and `REJECT`. An example of a call termination request:

```
POST /calls/4310001/terminate HTTP/1.1
Content-Type: application/json

{
  "actualMsisdn": 4369912345,
  "callingMsisdn": 4310001,
  "actualClir": 0,
  "callId": "abcd",
  "token": "PCC-aijfeoi",
  "reason": "BYE"
}
```

NGCP will not take the response of PCC server into consideration, because the call has already been terminated at SIP protocol level.

### 16.4.3 Voicemail Notification

#### 16.4.3.1 Using the PCC Framework

The PCC call control framework may also be used for voicemail notifications. The Sipwise NGCP involves its elements: *asterisk* (Voicemail server) and *ngcp-vmnotify* in the process of the notification.

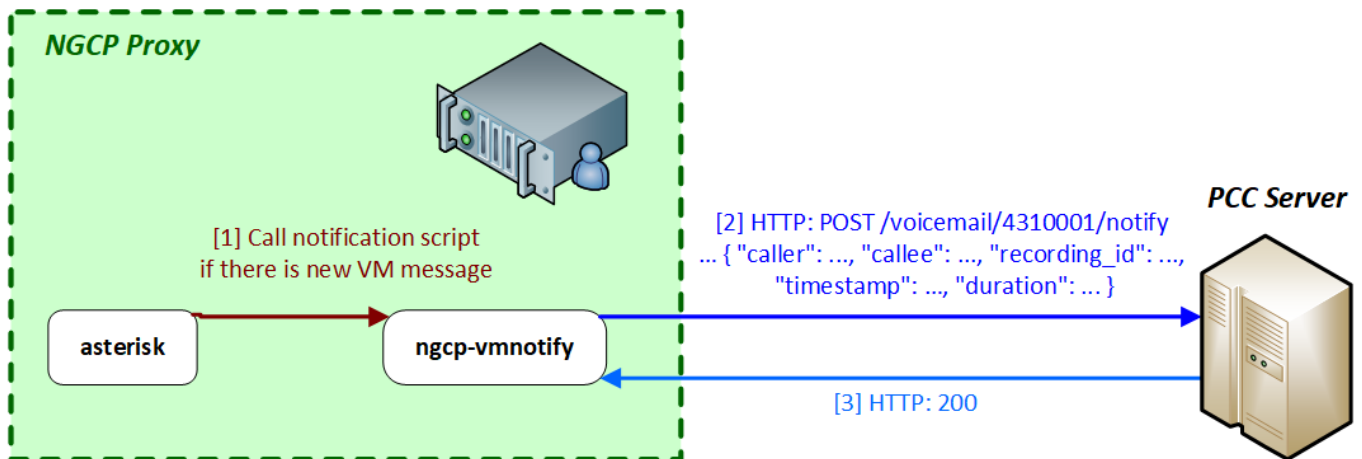


Figure 155: Voicemail Notification with PCC

1. The *asterisk* voicemail server triggers the *ngcp-vmnotify* script when a caller leaves a voicemail message in the callee's voicebox.
2. *ngcp-vmnotify* sends an HTTP *POST* request to the PCC server, as given in the example below:

```
POST /voicemail/4310001/notify HTTP/1.1
Content-Type: application/json

{
  "caller": 4369912345,
  "callee": 4310001,
  "recording_id": 45235 ,
  "timestamp": "2017-06-13T14:21:17T+01:00",
  "duration": 17
}
```

The target URL is: **/voicemail/<called\_party\_num>/notify**

3. The PCC server replies with *200 OK* if it properly processed the request.

### 16.4.3.2 Using SMS

The NGCP also supports voicemail notifications in form of short messages, using the built-in SMS modules. In such a case the *ngcp-vmnotify* module will send an HTTP *POST* request to the REST API (NGCP Web), that will contain the short message and finally be stored in the central database. Afterwards the short message will be sent to the recipient by NGCP Proxy.

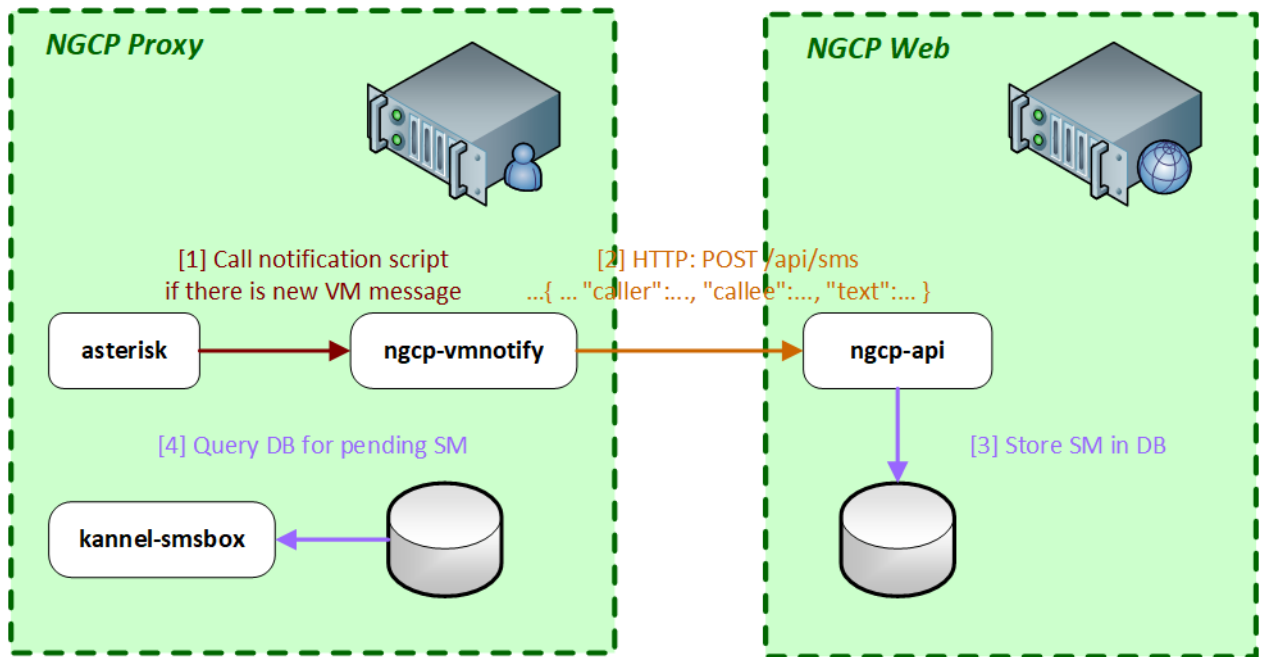


Figure 156: Voicemail Notification with SMS

1. The *asterisk* voicemail server triggers the *ngcp-vmnotify* script when a caller leaves a voicemail message in the callee's voicebox.
2. *ngcp-vmnotify* sends an API request to *ngcp-api* module, as given in the example below:

```
POST /api/sms/?skip_checks=true&skip_journal=false HTTP/1.1
Content-Type: application/json

{
  "subscriber_id": 90
  "caller": 4369912345,
  "callee" : 4310001,
  "text": "user1 4310001 17 Tue 13 Jun 2017 14:21:17 +01:00"
}
```

The target URL is: `/api/sms`

3. The *ngcp-api* stores the message in the database.
4. The *kannel-smsbox* module of NGCP Proxy will query the database for messages waiting for delivery and send the SM to its recipient through NGCP LB.

#### 16.4.4 Incoming Short Message Acceptance

##### 16.4.4.1 Indicating Incoming SM to PCC Server

The PCC server may also serve as a control point for incoming short messages. The Sipwise NGCP may indicate an incoming SM to the PCC server, which in turn must explicitly accept the message, so that the message will be forwarded to the recipient.

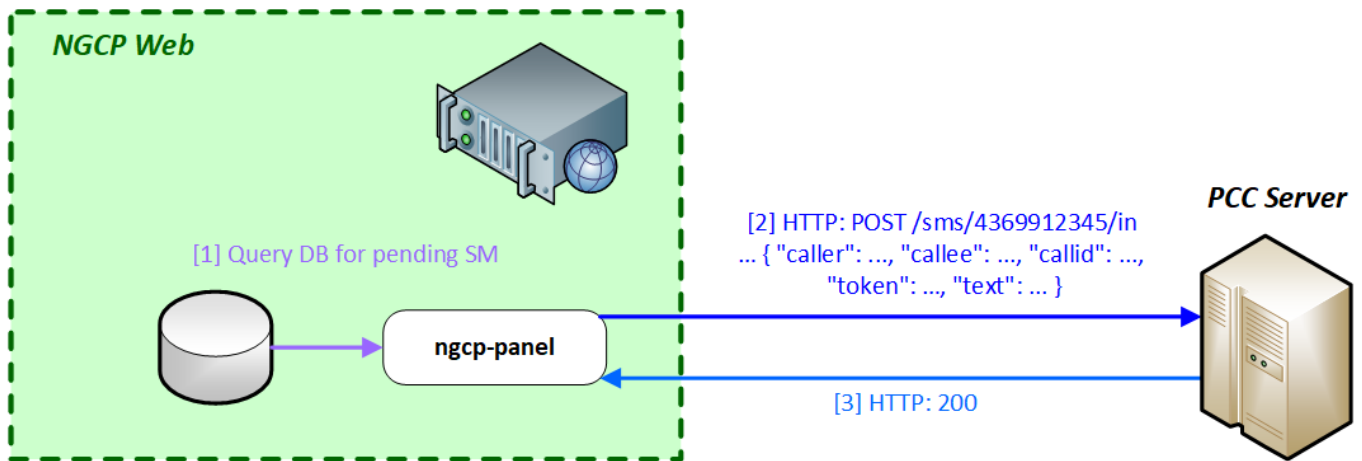


Figure 157: Short Message Notification with PCC

1. The `ngcp-panel` module on NGCP Web component will query the central database for pending incoming SMs.
2. The `ngcp-panel` will send an HTTP *POST* request to the PCC server if there is a message waiting for a subscriber. An example of such request is shown here:

```

POST /sms/4310001/in HTTP/1.1
Content-Type: application/json

{
  "caller": 4369912345,
  "callee": 4310001,
  "token": "PCC-aijfeoi",
  "callId": "abcd",
  "text": "This is the SM text"
}
  
```

The target URL in this case is: `/sms/<called_party_num>/in`

3. The PCC server replies with *200 OK* HTTP status if it properly understood the request.

#### 16.4.4.2 Incoming SM Accepted by PCC Server

As in the case of an incoming call, the PCC server will send an HTTP *POST* request to the REST API of NGCP, in order to signal the acceptance of the SM.

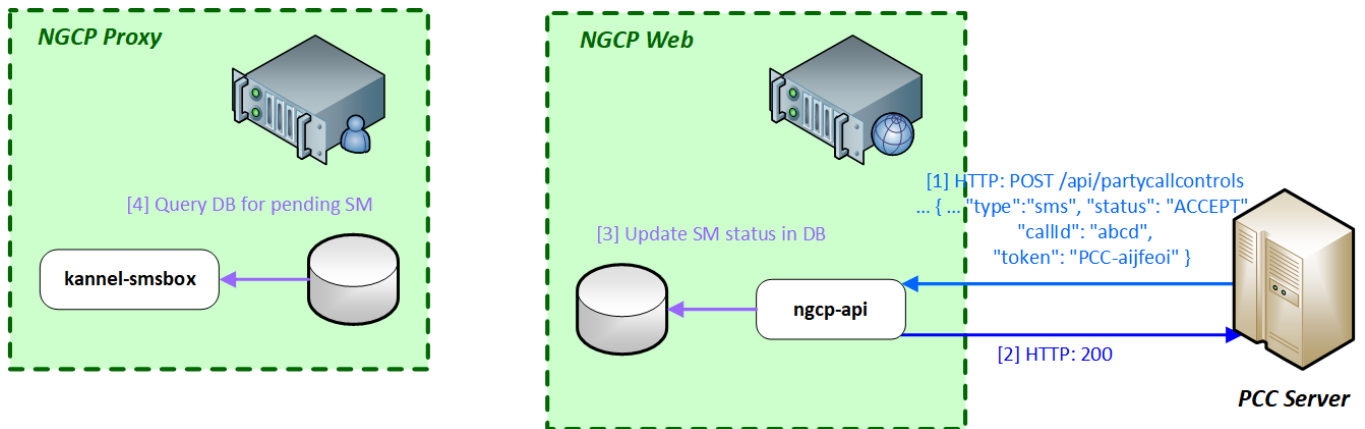


Figure 158: Short Message Accepted by PCC

1. The PCC server sends the request to NGCP Web component, where *ngcp-api* module will process it. An example:

```
POST /api/partycallcontrols HTTP/1.1
Content-Type: application/json

{
  "type": "sms",
  "caller": 4369912345,
  "callee": 4310001,
  "status": "ACCEPT",
  "callId": "abcd",
  "token": "PCC-aijfeoi"
}
```

The target URL of the request: `/api/partycallcontrols`. The **type** parameter must have a value of **sms**.

2. The *ngcp-api* module responds with `200 OK` HTTP status if it properly understood the request.
3. The *ngcp-api* updates the status of the SM in the database so that the SM may be forwarded to the recipient.
4. The *kannel-smsbox* module on NGCP Proxy will query the central database for SMs to be delivered and will forward the SM towards an SMSC, via NGCP LB.

### 16.4.5 Configuration of PCC

The configuration of the PCC feature is done via the main configuration file: `/etc/ngcp-config/config.yml`. The relevant section is: `apps.party_call_control`, the example below shows the default values of the parameters.

```
apps:
  party_call_control:
    accepted_reply: 200*
    enable: no
    pcc_server_url: https://127.0.0.1:9090/pcc/${prefix}${callee}${suffix}
    request_timeout: '30'
```

```
trigger_on_hangup: yes
```

The configuration parameters are:

- `accepted_reply`: defines the value of `status` data element (in the PCC server's `POST` request sent to `/api/party` `callcontrols` API resource) that means the "accepted" status of the call. For instance the handbook showed the value `ACCEPT` in previous sections, instead of the default `200*`
- `enable`: must be set to `yes` in order to enable the PCC feature
- `pcc_server_url`: the URL, pointing to the PCC server, where HTTP `POST` requests must be sent. The variables `${prefix}`, `${callee}` and `${suffix}` will be replaced with actual values when a request is sent. *Please do not change this part of the URL!* Possible values are:
  - `prefix = calls, suffix = initiate`
  - `prefix = calls, suffix = terminate`
  - `prefix = voicemail, suffix = notify`
  - `prefix = sms, suffix = in`
  - `callee = <called_party_num>`
- `request_timeout`: time in seconds until NGCP will wait for an HTTP reply from the PCC server, once the NGCP has sent a request to it
- `trigger_on_hangup`: if set to `yes`, NGCP will send a "terminate" request to the PCC server at the end of the call

#### 16.4.6 Troubleshooting of PCC

The Sipwise NGCP will provide logs of its activities that are very useful for troubleshooting the call processing with PCC feature. This section will provide examples from various log files that can help to find potential problems in call setup.

##### 16.4.6.1 Kamailio Proxy Log

###### PCC activation at `ngcp-sems` module

```
Oct 17 17:00:45 prx01a proxy[3206]: NOTICE: <script>: Call to PCC (Party Call Control) - R= ←
sip:2133339@192.168.10.11:5060;user=phone ID=1849964028_125696279@10.0.0.121 UA='<null>'
```

###### Call accepted by PCC server

```
Oct 17 17:00:16 prx01a proxy[3210]: NOTICE: <script>: NAT-Reply - S=301 - Accepted M=INVITE ←
IP=192.168.10.12:5080 (192.168.10.12:5080) ID=1850250074_83465152@10.0.0.121 UA='<null' ←
>'
Oct 17 17:00:16 prx01a proxy[3210]: INFO: <script>: Received 200 OK (Accepted) from PCC ←
Server, routing the call to its original callee - ID=1850250074_83465152@10.0.0.121 UA ←
='<null>'
```

### 16.4.6.2 SEMS Log

#### Initiate call at PCC

```
Oct 17 17:10:47 prx01a sems[5059]: [#7f73237f7700] [mod_py_log, PyDSM.cpp:42] INFO: PCC ↔
  http request to http://example.com/pcc/calls/4366811112222/initiate - callid 1851794724 ↔
  _134068006@10.0.0.121
Oct 17 17:10:47 prx01a sems[5059]: [#7f73237f7700] [mod_py_log, PyDSM.cpp:42] INFO: PCC ↔
  form data: {'actualMsisdn': '4369933334444', 'actualClir': '0', 'token': 'PCC-12DBBD25 ↔
  -59E61D770001841C-237F7700', 'callingMsisdn': '4366811112222', 'callId': '1851794724 ↔
  _134068006@10.0.0.121'} - callid 1851794724_134068006@10.0.0.121
Oct 17 17:10:47 prx01a sems[5059]: [#7f73237f7700] [mod_py_log, PyDSM.cpp:42] INFO: PCC ret ↔
  : 0 num_handles: 1
Oct 17 17:10:47 prx01a sems[5059]: [#7f73237f7700] [mod_py_log, PyDSM.cpp:42] INFO: RT: 0 1 ↔
  0 [] []
...
Oct 17 17:10:47 prx01a sems[5059]: [#7f73237f7700] [mod_py_log, PyDSM.cpp:42] INFO: RT: 0 0 ↔
  0 [<pycurl.Curl object at 0x7f7378067c50>] []
Oct 17 17:10:47 prx01a sems[5059]: [#7f73237f7700] [mod_py_log, PyDSM.cpp:42] INFO: PCC ↔
  reply for callid 1851794724_134068006@10.0.0.121: 200
```

#### Call accepted by PCC server

```
Oct 17 17:10:51 prx01a sems[5059]: [#7f7323efe700] [execute, XMLRPC2DI.cpp:714] INFO: ↔
  XMLRPC2DI 'postDSMEvent': function 'postDSMEvent'
Oct 17 17:10:51 prx01a sems[5059]: [#7f7323efe700] [execute, XMLRPC2DI.cpp:718] INFO: ↔
  params: <['PCC-12DBBD25-59E61D770001841C-237F7700', [['cmd', 'handleCall'], ['callid', ↔
  '1851794724_134068006@10.0.0.121'], ['caller', '4369933334444'], ['callee', ↔
  '4366811112222'], ['status', 'ACCEPT']]>
Oct 17 17:10:51 prx01a sems[5059]: [#7f7323efe700] [execute, XMLRPC2DI.cpp:724] INFO: ↔
  result: <[200, 'OK']>
Oct 17 17:10:51 prx01a sems[5059]: [#7f73237f7700] [execute, DSMCoreModule.cpp:521] INFO: ↔
  FSM: 'PCC RESULT -- ACCEPT'
```

#### Terminate call at PCC

```
Oct 17 17:10:53 prx01a sems[5059]: [#7f73235f5700] [mod_py_log, PyDSM.cpp:42] INFO: PCC ↔
  http request to http://example.com/pcc/calls/4366811112222/terminate - callid 1851794724 ↔
  _134068006@10.0.0.121
Oct 17 17:10:53 prx01a sems[5059]: [#7f73235f5700] [mod_py_log, PyDSM.cpp:42] INFO: PCC ↔
  form data: {'actualMsisdn': '4369933334444', 'callId': '1851794724_134068006@10 ↔
  .0.0.121', 'callingMsisdn': '4366811112222', 'reason': 'CANCEL', 'token': 'PCC-12DBBD25 ↔
  -59E61D770001841C-237F7700', 'actualClir': '0'} - callid 1851794724_134068006@10.0.0.121
```

### 16.4.6.3 NGCP Panel Log

#### SM notification at PCC server

```

Oct 18 09:10:16 web01a ngcp-panel: INFO: pcc is set to 1 for prov subscriber id 18451
Oct 18 09:10:16 web01a ngcp-panel: INFO: >>>> source check for booking.com passed, continue ←
    with time check
Oct 18 09:10:16 web01a ngcp-panel: INFO: >>>> time check for 1508310615 passed, use ←
    destination set
Oct 18 09:10:16 web01a ngcp-panel: INFO: >>>> proceed sms forwarding
Oct 18 09:10:16 web01a ngcp-panel: INFO: >>>> forward sms to 4369933334444
Oct 18 09:10:16 web01a ngcp-panel: INFO: sending pcc request for sms with id 305125 to http ←
    ://example.com/pcc/sms/4366811112222/in
Oct 18 09:10:16 web01a ngcp-panel: INFO: sending pcc request succeeded
Oct 18 09:10:16 web01a ngcp-panel: INFO: status for pcc sms of 305125 is BUSY, don't ←
    forward sms

```

In the last line the status is BUSY. The purpose of this is to prevent forwarding the SM to the mobile phone of the recipient. Otherwise, in order to let NGCP forward the message to the recipient, the status is ACCEPT.

#### 16.4.6.4 REST API Log

##### Call accepted by PCC server

```

Oct 18 10:19:39 web01a ngcp-panel: INFO: IP=192.168.10.20 CALLED=API[POST]/api/ ←
    partycallcontrols/ TX=14EE9C4CD2599A70 USER=username DATA={} MSG="" LOG="{ "type": "pcc", " ←
    caller": "4365033334444", "callee": "4366811112222", "status": "ACCEPT", "token": "PCC-273C2CDA ←
    -59E70E9600BEOC4-231F1700", "callid": "406885946_117428858@10.0.0.121"}"
Oct 18 10:19:39 web01a ngcp-panel: INFO: IP=192.168.10.20 CALLED=API[POST 200]/api/ ←
    partycallcontrols/ TX=14EE9C4CD2599A70 USER=username DATA={} MSG="" LOG=""

```

##### SM accepted by PCC server

```

Oct 18 10:20:30 web01a ngcp-panel: INFO: IP=192.168.10.20 CALLED=API[POST]/api/ ←
    partycallcontrols/ TX=14EE9C58CEA4D960 USER=username DATA={} MSG="" LOG="{ "type": "sms", " ←
    caller": "15556666", "callee": "4366811112222", "status": "ACCEPT", "token": "1482d9e2-a9fc-40 ←
    ee-bdaf-de6f7fc239f8", "callid": "305175"}"
Oct 18 10:20:30 web01a ngcp-panel: INFO: IP=192.168.10.20 CALLED=API[POST 200]/api/ ←
    partycallcontrols/ TX=14EE9C58CEA4D960 USER=username DATA={} MSG="" LOG=""

```

#### 16.4.6.5 Voicemail Notification Log

The voicemail notifier script (/usr/bin/vmnotify) writes its log messages into the system log (/var/log/syslog). An example:

```

Oct 18 09:53:34 prx01a vmnotify[20072]: Arguments: default 4366811112222 1 0 0 0 ←
    4365033334444 2017-10-18T09:53:34+0200 8

```

Where the *Arguments* are:



- default: Asterisk voicemail context
- the voicemail box owner
- 1: number of new messages
- 0: number of old messages
- 0: number of urgent messages
- 0: message ID of the latest message
- who left the message (caller)
- date and time of the message
- 8: duration of the message in seconds

## A Basic Call Flows

### A.1 General Call Setup

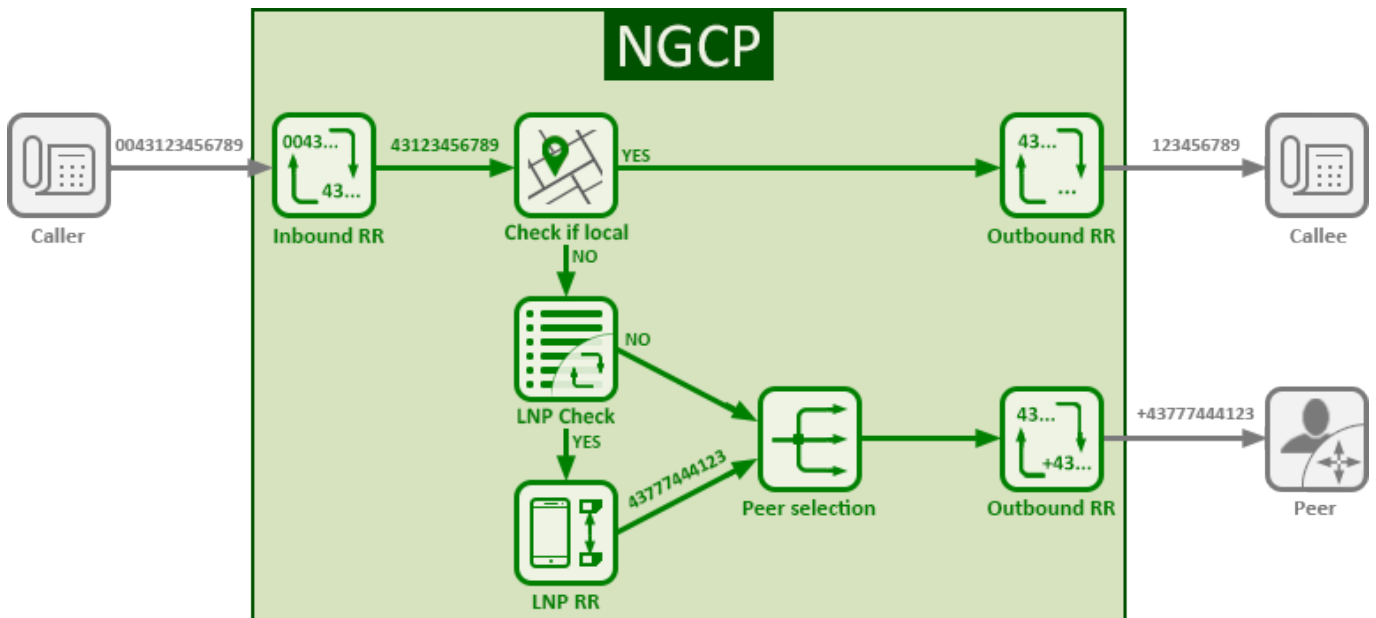


Figure 159: General Call Setup

NGCP performs the following checks when processing a call coming from a subscriber and terminated at a peer:

- Checks if the IP address where the request came from is in the list of trusted IP addresses. If yes, this IP address is taken as the identity for authentication. Otherwise, NGCP performs the digest authentication.
- When the subscriber is authorized to make the call, NGCP applies the Inbound Rewrite Rules for the caller and the callee assigned to the subscriber (if any). If there are no Rewrite Rules assigned to the subscriber, the ones assigned to the subscriber's domain are applied. On this stage the platform normalises the numbers from the subscriber's format to E.164.
- Matches the callee (called number) with local subscribers.
  - If it finds a matching subscriber, the call is routed internally. In this case, NGCP applies the Outbound Rewrite Rules associated with the callee (if any). If there are no Rewrite Rules assigned to the callee, the ones assigned to the callee's domain are applied.
  - If it does not find a matching subscriber, the call goes to a peer as described below.
- Queries the LNP database to find out if the number was ported or not. For details of LNP queries refer to the [Local Number Porting](#) Section 5.4 chapter.
  - If it was ported, NGCP applies the LNP Rewrite Rules to the called number.
- Based on the priorities of peering groups and peering rules (see Section 4.6.2.3 for details), NGCP selects peering groups for call termination and defines their precedence.

- Within every peering group the weight of a peering server defines its probability to receive the call for termination. Thus, the bigger the weight of a server, the higher the probability that NGCP will send the call to it.
- Applies the Outbound Rewrite Rules for the caller and the callee assigned to a peering server when sending the call to it.

## A.2 Endpoint Registration

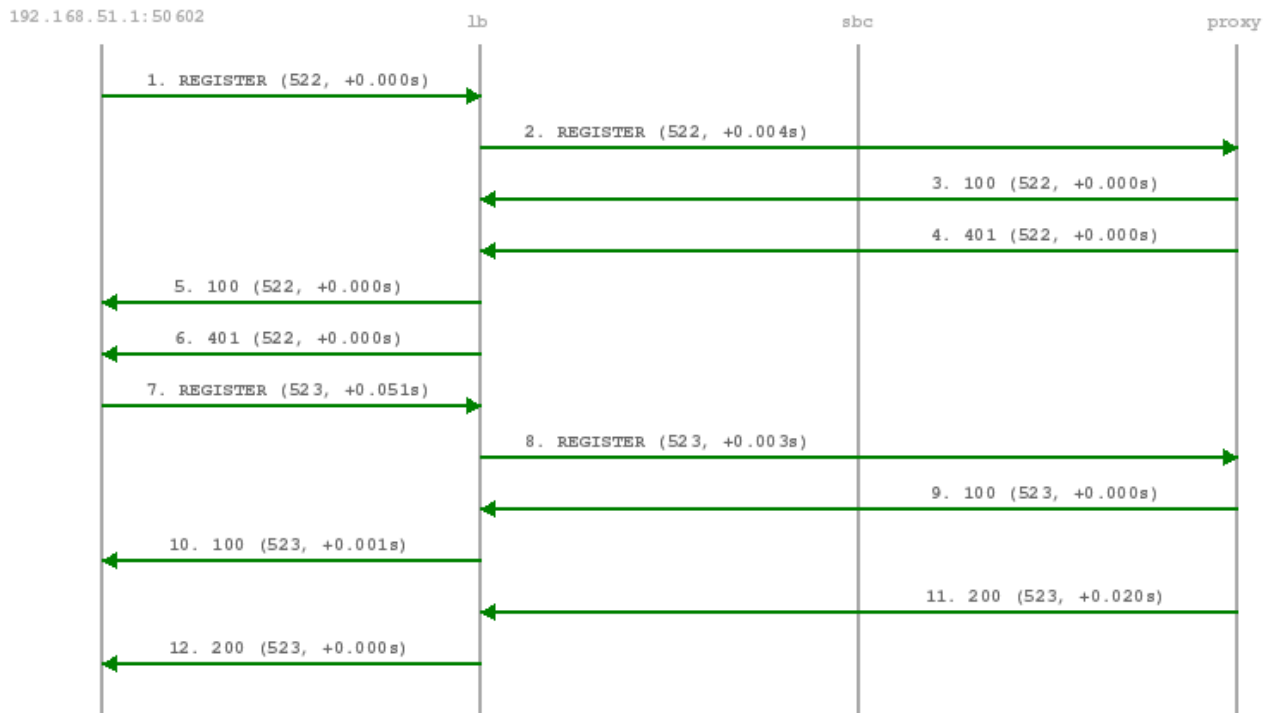


Figure 160: Registration Call-Flow

The subscriber endpoint starts sending a REGISTER request, which gets challenged by a 401. After calculating the response of the authentication challenge, it sends the REGISTER again, including the authentication response. The SIP proxy looks up the credentials of the subscriber in the database, does the same calculation, and if the result matches the one from the subscriber, the registration is granted.

The SIP proxy writes the content of the Contact header (e.g. sip:me@1.2.3.4:1234;transport=UDP) into its location table (in case of NAT the content is changed by the SIP load-balancer to the IP/port from where the request was received), so it knows where to reach a subscriber in case of an inbound call to this subscriber (e.g. sip:someuser@example.org is mapped to sip:me@1.2.3.4:1234;transport=UDP and sent out to this address).

If NAT is detected, the SIP proxy sends a OPTION message to the registered contact every 30 seconds, in order to keep the NAT binding on the NAT device open. Otherwise, for subsequent calls to this contact, the sip:provider PRO wouldn't be able to reach the endpoint behind NAT (NAT devices usually drop a UDP binding after not receiving any traffic for ~30-60 seconds).

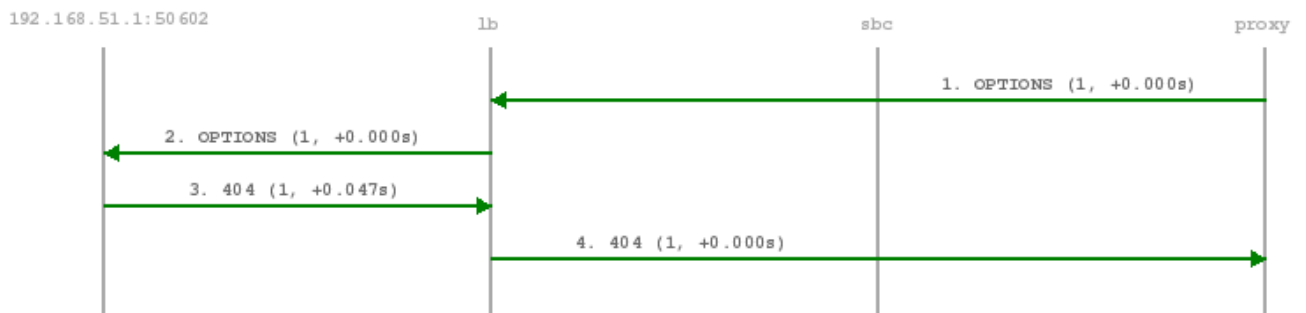


Figure 161: NAT-Ping Call-Flow

By default, a subscriber can register 5 contacts for an Address of Record (AoR, e.g. `sip:someuser@example.org`).



### A.3 Basic Call

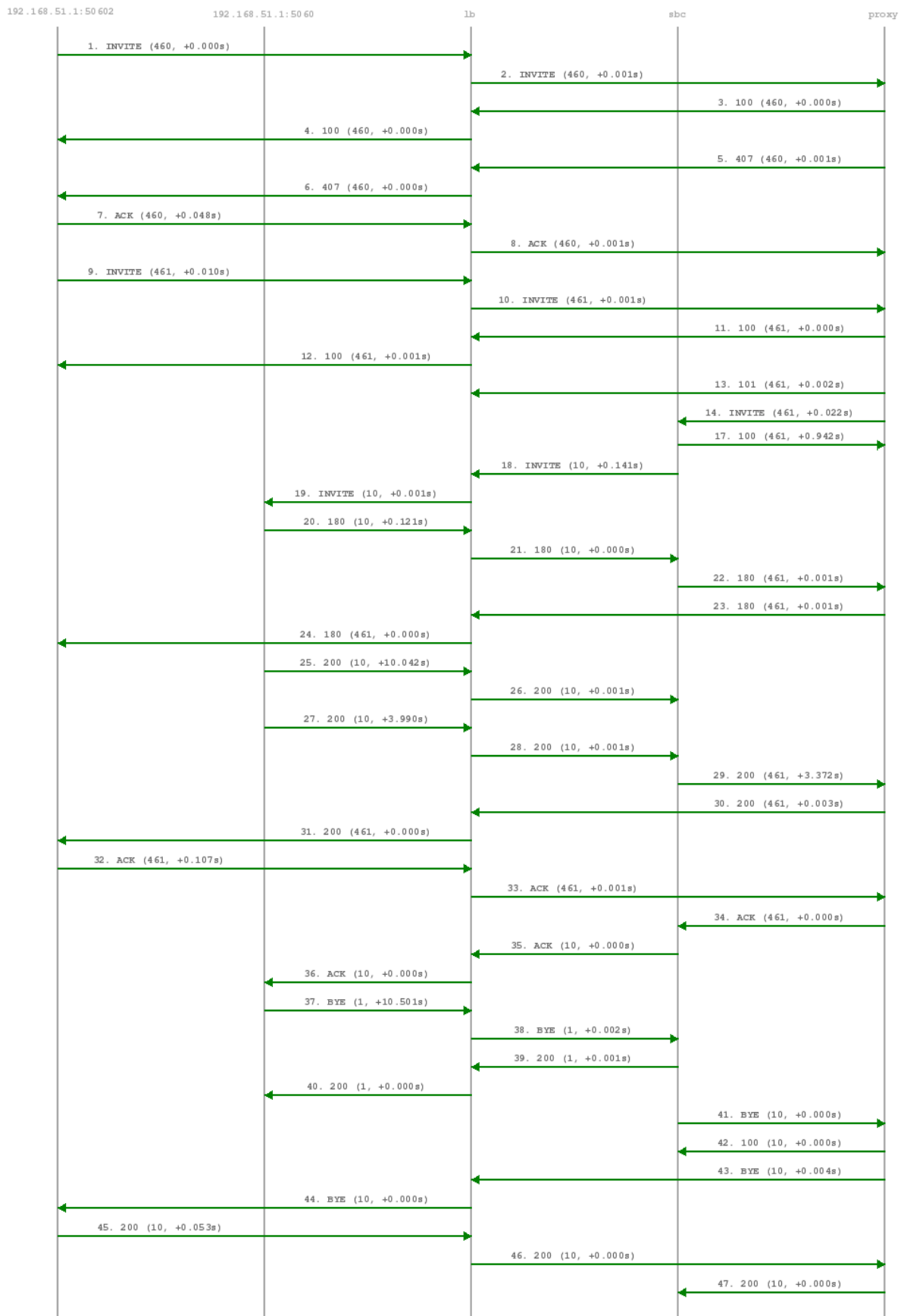


Figure 162: Basic Call Call-Flow

The calling party sends an INVITE (e.g. `sip:someuser@example.org`) via the SIP load-balancer to the SIP proxy. The proxy replies with an authorization challenge in the 407 response, and the calling party sends the INVITE again with authentication credentials. The SIP proxy checks if the called party is a local user. If it is, and if there is a registered contact found for this user, then (after various feature-related tasks for both the caller and the callee) the Request-URI is replaced by the URI of the registered contact (e.g. `sip:me@1.2.3.4:1234;transport=UDP`). If it's not a local user but a numeric user, a proper PSTN gateway is being selected by the SIP proxy, and the Request-URI is rewritten accordingly (e.g. `sip:+43123456789@2.3.4.5:5060`).

Once the proxy has finished working through the call features of both parties involved and has selected the final destination for the call, and - optionally - has invoked the Media Relay for this call, the INVITE is sent to the SIP B2BUA. The B2BUA creates a new INVITE message from scratch (using a new Call-ID and a new From-Tag), copies only various and explicitly allowed SIP headers from the old message to the new one, filters out unwanted media capabilities from the SDP body (e.g. to force audio calls to use G.711 as a codec) and then sends the new message via the SIP load-balancer to the called party.

SIP replies from the called party are passed through the elements back to the calling party (replacing various fields on the B2BUA to match the first call leg again). If a reply with an SDP body is received by the SIP proxy (e.g. a 183 or a 200), the Media Relay is invoked again to prepare the ports for the media stream.

Once the 200 is routed from the called party to the calling party, the media stream is fully negotiated, and the endpoints can start sending traffic to each other (either end-to-end or via the Media Relay). Upon reception of the 200, the SIP proxy writes a start record for the accounting process. The 200 is also acknowledged with an ACK message from the calling party to the called party, according to the SIP 3-way handshake.

Either of the parties can tear down the media session at any time by sending a BYE, which is passed through to the other party. Once the BYE reaches the SIP proxy, it instructs the Media Relay to close the media ports, and it writes a stop record for accounting purposes. Both the start- and the stop-records are picked up by the *mediator* service in a regular interval and are converted into a Call Detail Record (CDR), which will be rated by the *rate-o-mat* process and can be billed to the calling party.

## A.4 Session Keep-Alive

The SIP B2BUA acts as refresher for the Session-Timer mechanism as defined in RFC 4028. If the endpoints indicate support for the UPDATE method during call-setup, then the SIP B2BUA will use an UPDATE message if enabled per peer, domain or subscriber via Provisioning to check if the endpoints are still alive and responsive. Both endpoints can renegotiate the timer within a configurable range. All values can be tuned using the Admin Panel or the APIs using Peer-, Domain- and Subscriber-Preferences.

---

### Tip

Keep in mind that the values being used in the signaling are always half the value being configured. So if you want to send a keep-alive every 300 seconds, you need to provision `sst_expires` to 600.

---

If one of the endpoints doesn't respond to the keep-alive messages or answers with `481 Call/Transaction Does Not Exist`, then the call is torn down on both sides. This mechanism prevents excessive over-billing of calls if one of the endpoints is not reachable anymore or "forgets" about the call. The BYE message sent by the B2BUA triggers a stop-record for accounting and also closes the media ports on the Media Relay to stop the call.

Beside the Session-Timer mechanism to prevent calls from being lost or kept open, there is a **maximum call length** of 21600 seconds per default defined in the B2BUA. This is a security/anti-fraud mechanism to prevent overly long calls causing excessive costs.

### A.5 Voicebox Calls

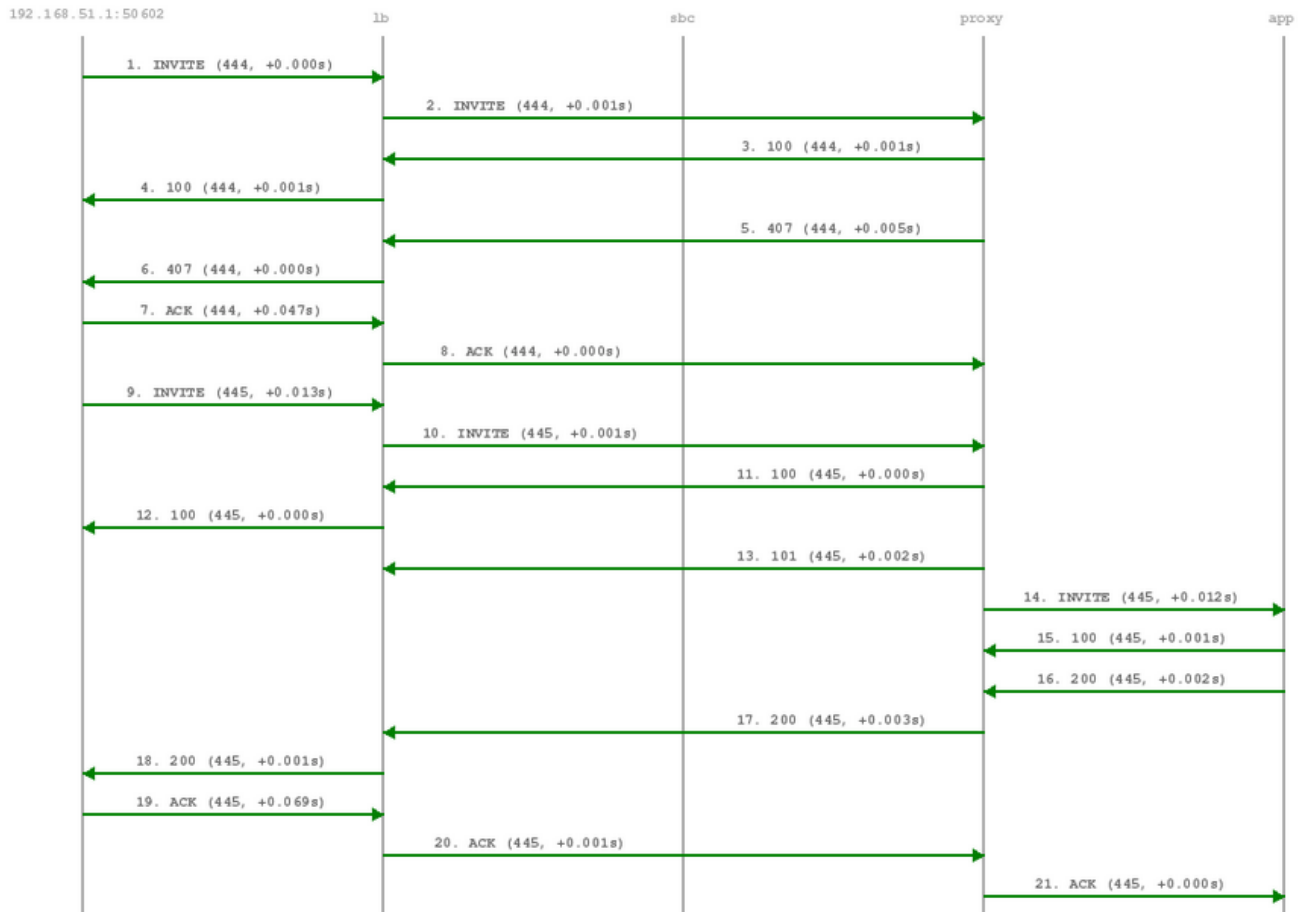


Figure 163: Voicebox Call-Flow

Calls to the Voicebox (both for callers leaving a voicemail message and for voicebox owners managing it via the IVR menu) are passed directly from the SIP proxy to the App-Server without a B2BUA. The App-Server maintains its own timers, so there is no risk of over-billing or overly long calls.

In such a case where an endpoint talks via the Media Relay to a system-internal endpoint, the Media Relay bridges the media streams between the public in the system-internal network.

In case of an endpoint leaving a new message on the voicebox, the Message-Waiting-Indication (MWI) mechanism triggers the sending of a unsolicited NOTIFY message, passing the number of new messages in the body. As soon as the voicebox owner dials into his voicebox (e.g. by calling `sip:voicebox@example.org` from his SIP account), another NOTIFY message is sent to his devices, resetting the number of new messages.



**Important**

The sip:carrier does not require your device to subscribe to the MWI service by sending a SUBSCRIBE (it would rather reject it). On the other hand, the endpoints need to accept unsolicited NOTIFY messages (that is, a NOTIFY without a valid subscription), otherwise the MWI service will not work with these endpoints.

---

## B NGCP configs overview

### B.1 config.yml Overview

`/etc/ngcp-config/config.yml` is the main configuration YAML file used by Sipwise NGCP. After every changes it need to run the command `ngcpcfg apply "my commit message"` to apply changes (followed by `ngcpcfg push` in the PRO version to apply changes to sp2). The following is a brief description of the main variables contained into `/etc/ngcp-config/config.yml` file.

#### B.1.1 apps

This section contains parameters for the additional applications that may be activated on sip:carrier.

```
apps:
  malicious_call: no
  party_call_control:
    accepted_reply: 200*
    enable: no
    pcc_server_url: https://127.0.0.1:9090/pcc/${prefix}${callee}${suffix}
    request_timeout: '30'
    trigger_on_hangup: yes
```

- `malicious_call`: If set to `yes`, the Malicious Call Identification (MCID) application will be enabled.
- `party_call_control.accepted_reply`: Defines the value of `status` data element that means the "accepted" status of the call.
- `party_call_control.enable`: Must be set to `yes` in order to enable the PCC feature.
- `party_call_control.pcc_server_url`: The URL, pointing to the PCC server, where HTTP `POST` requests must be sent. Do not change the variable references `${prefix}`, `${callee}` and `${suffix}`!
- `party_call_control.request_timeout`: Time in seconds until NGCP will wait for an HTTP reply from the PCC server, once the NGCP has sent a request to it.
- `party_call_control.trigger_on_hangup`: If set to `yes`, NGCP will send a "terminate" request to the PCC server at the end of the call.

---

#### Tip

See the Section [16.4.5](#) section of the handbook for more details on PCC configuration.

---

#### B.1.2 asterisk

The following is the asterisk section:

```

asterisk:
  log:
    facility: local6
  rtp:
    maxport: 20000
    minport: 10000
  sip:
    bindport: 5070
    dtmfmode: rfc2833
  voicemail:
    enable: 'no'
    fromstring: 'Voicemail server'
    greeting:
      busy_custom_greeting: '/home/user/file_no_extension'
      busy_overwrite_default: 'no'
      busy_overwrite_subscriber: 'no'
      unavail_custom_greeting: '/home/user/file_no_extension'
      unavail_overwrite_default: 'no'
      unavail_overwrite_subscriber: 'no'
    mailbody: 'You have received a new message from ${VM_CALLERID} in voicebox ${VM_MAILBOX} ←
      } on ${VM_DATE}.'
    mailsubject: '[Voicebox] New message ${VM_MSGNUM} in voicebox ${VM_MAILBOX}'
    max_msg_length: 180
    maxgreet: 60
    maxmsg: 30
    maxsilence: 0
    min_msg_length: 3
    normalize_match: '^00|\+([1-9][0-9]+)$'
    normalize_replace: '$1'
    serveremail: voicebox@sip.sipwise.com

```

- `log.facility`: rsyslog facility for asterisk log, defined in `/etc/asterisk/logger.conf`.
- `rtp.maxport`: RTP maximum port used by asterisk.
- `rtp.minport`: RTP minimum port used by asterisk.
- `sip.bindport`: SIP asterisk internal bindport.
- `voicemail.greetings.*`: set the audio file path for voicemail custom unavailable/busy greetings
- `voicemail.mailbody`: Mail body for incoming voicemail.
- `voicemail.mailsubject`: Mail subject for incoming voicemail.
- `voicemail.max_msg_length`: Sets the maximum length of a voicemail message, in seconds.
- `voicemail.maxgreet`: Sets the maximum length of voicemail greetings, in seconds.
- `voicemail.maxmsg`: Sets the maximum number of messages that may be kept in any voicemail folder.

- `voicemail.min_msg_length`: Sets the minimum length of a voicemail message, in seconds.
- `voicemail.maxsilence`: Maxsilence defines how long Asterisk will wait for a contiguous period of silence before terminating an incoming call to voice mail. The default value is 0, which means the silence detector is disabled and the wait time is infinite.
- `voicemail.serveremail`: Provides the email address from which voicemail notifications should be sent.
- `voicemail.normalize_match`: Regular expression to match the From number for calls to voicebox.
- `voicemail.normalize_replace`: Replacement string to return, in order to match an existing voicebox.

### B.1.3 autoprov

The following is the autoprovisioning section:

```
autoprov:
  hardphone:
    skip_vendor_redirect: 'no'
  server:
    bootstrap_port: 1445
    ca_certfile: '/etc/ngcp-config/ssl/client-auth-ca.crt'
    host: localhost
    port: 1444
    server_certfile: '/etc/ngcp-config/ssl/myserver.crt'
    server_keyfile: '/etc/ngcp-config/ssl/myserver.key'
    ssl_enabled: 'yes'
  softphone:
    config_lockdown: 0
    webauth: 0
```

- `autoprov.skip_vendor_redirect`: Skip phone vendor redirection to the vendor provisioning web site.

### B.1.4 backuptools

The following is the backup tools section:

```
backuptools:
  cdreexport_backup:
    enable: 'no'
  etc_backup:
    enable: 'no'
  mail:
    address: noc@company.org
    error_subject: '[ngcp-backup] Problems detected during daily backup'
    log_subject: '[ngcp-backup] Daily backup report'
    send_errors: 'no'
    send_log: 'no'
  mysql_backup:
```

```
enable: 'no'
exclude_dbs: 'syslog sipstats information_schema'
rotate_days: 7
storage_dir: '/var/backup/ngcp_backup'
temp_backup_dir: '/tmp/ngcp_backup'
```

- `backuptools.cdrexport_backup.enable`: Enable backup of cdrexport (.csv) directory.
- `backuptools.etc_backup.enable`: Enable backup of /etc/\* directory.
- `backuptools.mail.address`: Destination email address for backup emails.
- `backuptools.mail.error_subject`: Subject for error emails.
- `backuptools.mail.log_subjetc`: Subject for daily backup report.
- `backuptools.mail.send_error`: Send daily backup error report.
- `backuptools.mail.send_log`: Send daily backup log report.
- `backuptools.mysql_backup.enable`: Enable daily mysql backup.
- `backuptools.mysql_backup.exclude_dbs`: exclude mysql databases from backup.
- `backuptools.rotate_days`: Number of days backup files should be kept. All files older than specified number of days are deleted from the storage directory.
- `backuptools.storage_dir`: Storage directory of backups.
- `backuptools.storage_group`: Name of the group that backup files should be owned by.
- `backuptools.storage_user`: Name of the user that backup files should be owned by.
- `backuptools.temp_backup_dir`: Temporary storage directory of backups.

### B.1.5 bootenv

The following is the bootenv section:

```
bootenv:
  dhcp:
    boot: '/srv/tftp/pxelinux.0'
    enable: 'yes'
    end: 192.168.1.199
    expire: 12h
    start: 192.168.1.101
  http_port: 3000
  http_proxy: ''
  https_proxy: ''
  ro_port: 9998
  rw_port: 9999
```

```
tftp:
  enable: 'yes'
  root: '/srv/tftp'
```

- `bootenv.dhcp.enable`: enable dnsmasq DHCP server
- `bootenv.dhcp.boot`: PXE image boot location
- `bootenv.dhcp.start`: first IP of DHCP scope
- `bootenv.dhcp.end`: last IP of DHCP scope
- `bootenv.dhcp.expire`: DHCP leasing expiration
- `bootenv.http_port`: HTTP port for iPXE boot files/configs
- `bootenv.http_proxy`: HTTP proxy to access Sipwise Debian repositories
- `bootenv.https_proxy`: HTTPS proxy to access Sipwise Debian repositories
- `bootenv.ro_port`: HTTP port for read-only access to Approx cache
- `bootenv.rw_port`: HTTP port for read-write access to Approx cache
- `bootenv.tftp.enable`: enable tftp server for PXE boot
- `bootenv.tftp.root`: root folder for tftp server

### B.1.6 cdreexport

The following is the cdr export section:

```
cdreexport:
  daily_folder: 'yes'
  export_failed: 'no'
  export_incoming: 'no'
  exportpath: '/home/jail/home/cdreexport'
  full_names: 'yes'
  monthly_folder: 'yes'
```

- `cdreexport.daily_folder`: Set *yes* if you want to create a daily folder for CDRs under the configured path.
- `cdreexport.export_failed`: Export CDR for failed calls.
- `cdreexport.export_incoming`: Export CDR for incoming calls.
- `cdreexport.exportpath`: The path to store CDRs in .csv format.
- `cdreexport.full_names`: Use full namen for CDRs instead of short ones.
- `cdreexport.monthly_folder`: Set *yes* if you want to create a monthly folder (ex. 201301 for January 2013) for CDRs under configured path.

## B.1.7 checktools

The following is the check tools section:

```
checktools:
  active_check_enable: '1'
  asr_ner_statistics: '1'
  collcheck:
    cpuidle: '0.1'
    dfused: '0.9'
    eximmaxqueue: '15'
    kamminshmem: '1048576'
    lbminshmem: '1048576'
    loadlong: '2'
    loadmedium: '2'
    loadshort: '3'
    maxage: 30
    memused: 0.98
    siptimeout: '15'
    sslcert_timetoexpiry: '30'
    sslcert_whitelist: []
    swapfree: 0.02
  exim_check_enable: '1'
  force: '0'
  kamailio_check_concurrent_calls_enable: '1'
  kamailio_check_dialog_active_enable: '1'
  kamailio_check_dialog_early_enable: '1'
  kamailio_check_dialog_incoming_enable: '1'
  kamailio_check_dialog_local_enable: '1'
  kamailio_check_dialog_outgoing_enable: '1'
  kamailio_check_dialog_relay_enable: '1'
  kamailio_check_shmem_enable: '1'
  kamailio_check_usrloc_regdevices_enable: '1'
  kamailio_check_usrloc_regusers_enable: '1'
  monitor_peering_groups: '1'
  mpt_check_enable: '0'
  mysql_check_enable: '1'
  mysql_check_replication: '1'
  mysql_replicate_check_interval: '3600'
  mysql_replicate_check_tables:
    - accounting
    - billing
    - carrier
    - kamailio
    - ngcp
    - provisioning
    - prosody
    - rtcengine
```

```

- stats
mysql_replicate_ignore_tables:
- accounting.acc_backup
- accounting.acc_trash
- kamilio.acc_backup
- kamilio.acc_trash
- ngcp.pt_checksums_sp1
- ngcp.pt_checksums_sp2
- ngcp.pt_checksums
oss_check_provisioned_subscribers_enable: '1'
sip_check_enable: '1'
sipstats_check_num_packets: '1'
sipstats_check_num_packets_perday: '1'
sipstats_check_partition_size: '1'
snmpd:
  communities:
    public:
      - localhost
  trap_communities:
    public:
      - localhost

```

- `checktools.collcheck.cpuidle`: Sets the minimum value for CPU usage (0.1 means 10%).
- `checktools.collcheck.dfused`: Sets the maximum value for DISK usage (0.9 means 90%).
- `checktools.collcheck.loadlong/loadlong/loadshort`: Max values for load (long, short, medium term).
- `checktools.collcheck.maxage`: Max age in seconds.
- `checktools.collcheck.memused`: Sets the maximum value for MEM usage (0.7 means 70%).
- `checktools.collcheck.siptimeout`: Max timeout for sip options.
- `checktools.collcheck.swapfree`: Sets the minimum value for SWAP free (0.5 means 50%).
- `checktools.exim_check_enable`: Exim queue check plugin for *ngcp-witnessd*.
- `checktools.active_check_enable`: Active node check plugin for *ngcp-witnessd*.
- `checktools.asr_ner_statistics`: enable/Disable ASR/NER statistics.
- `checktools.force`: Perform checks even if not active from `ngcp-check-active` command.
- `checktools.kamilio_check_*`: Enable/Disable SNMP collective check plugin for Kamailio.
- `checktools.mpt_check_enable`: MPT raid SNMP check plugin.
- `checktools.mysql_check_enable`: Enable/disable MySQL check SNMP plugin.
- `checktools.mysql_check_replication`: Enable/disable MySQL replication check.



- `checktools.mysql_replicate_check_interval`: MySQL replication check interval in seconds.
- `checktools.mysql_replicate_check_tables`: List of tables that need to be checked for replication issues.
- `checktools.mysql_replicate_ignore_tables`: List of tables that need to be ignored during replication check.
- `checktools.oss_check_provisioned_subscribers_enable`: OSS provisioned subscribers count plugin.
- `checktools.sip_check_enable/sipstats_check_*`: Enable/Disable SIP check plugins.
- `checktools.snmpd.communities.*`: Sets the SNMP community and sources. Entries (i.e. the *sources*) under a community (like `public` in the example) are in a list format, each line starting with "-" and followed by the source address.
- `checktools.snmpd.trap_communities.*`: Sets the SNMP TRAP community and destination for traps sent by NGCP. Format is the same as for `checktools.snmpd.communities`.

### B.1.8 cleanuptools

The following is the cleanup tools section:

```

cleanuptools:
  acc_cleanup_days: 90
  archive_targetdir: '/var/backups/cdr'
  binlog_days: 15
  cdr_archive_months: 2
  cdr_backup_months: 2
  cdr_backup_retro: 3
  compress: gzip
  delete_old_cdr_files:
    enabled: 'no'
    max_age_days: 30
    paths:
      -
        max_age_days: ~
        path: '/home/jail/home/*/20[0-9][0-9][0-9][0-9]/[0-9][0-9]'
        remove_empty_directories: 'yes'
        wildcard: 'yes'
      -
        max_age_days: ~
        path: '/home/jail/home/cdrexport/resellers/*/20[0-9][0-9][0-9][0-9]/[0-9][0-9]'
        remove_empty_directories: 'yes'
        wildcard: 'yes'
      -
        max_age_days: ~
        path: '/home/jail/home/cdrexport/system/20[0-9][0-9][0-9][0-9]/[0-9][0-9]'
        remove_empty_directories: 'yes'
        wildcard: 'yes'
  sql_batch: 10000
  trash_cleanup_days: 30

```

- `cleantools.acc_cleanup_days`: CDR records in `acc` table in `kamailio` database will be deleted after this time
- `cleantools.binlog_days`: Time after MySQL binlogs will be deleted.
- `cleantools.cdr_archive_months`: How many months worth of records to keep in monthly CDR backup tables, instead of dumping them into archive files and dropping them from database.
- `cleantools.cdr_backup_months`: How many months worth of records to keep in the current `cdr` table, instead of moving them into the monthly CDR backup tables.
- `cleantools.cdr_backup_retro`: How many months to process for backups, going backwards in time and skipping `cdr_backup_months` months first, and store them in backup tables. Any older record will be left untouched.
- `cleantools.delete_old_cdr_files`:
  - `enabled`: Enable (`yes`) or disable (`no`) exported CDR cleanup.
  - `max_age_days`: Gives the expiration time of the exported CDR files in days. There is a general value which may be overridden by a local value provided at a specific path. The local value is valid for the particular path only.
  - `paths`: an array of path definitions
    - \* `path`: a path where CDR files are to be found and deleted; this may contain wildcard characters
    - \* `wildcard`: Enable (`yes`) or disable (`no`) using wildcards in the `path`
    - \* `remove_empty_directories`: Enable (`yes`) or disable (`no`) removing empty directories if those are found in the given `path`
    - \* `max_age_days`: the local expiration time value for files in the particular `path`
- `cleantools.sql_batch`: How many records to process within a single SQL statement.
- `cleantools.trash_cleanup_days`: Time after CDRs from `acc_trash` and `acc_backup` tables in `kamailio` database will be deleted.

For the description of `cleantools` please visit [Cleanuptools Description](#) Section 13.4 section of the handbook.

### B.1.9 cluster\_sets

The following is the cluster sets section:

```
cluster_sets:
  default:
    dispatcher_id: 50
  default_set: default
  type: central
```

- `cluster_sets.<label>`: an arbitrary label of the cluster set; in the above example we have `default`
- `cluster_sets.<label>.dispatcher_id`: a unique, numeric value that identifies a particular cluster set
- `cluster_sets.default_set`: selects the default cluster set
- `cluster_sets.type`: the type of cluster set; can be `central` or `distributed`

### B.1.10 database

The following is the database section:

```
database:
  bufferpoolsize: 24768M
```

- `database.bufferpoolsize`: Innodb\_buffer\_pool\_size value in `/etc/mysql/my.cnf`

### B.1.11 faxserver

The following is the fax server section:

```
faxserver:
  enable: yes
  fail_attempts: '3'
  fail_retry_secs: '60'
  mail_from: 'Sipwise NGCP FaxServer <voipfax@ngcp.sipwise.local>'
```

- `faxserver.enable`: *yes/no* to enable or disable `ngcp-faxserver` on the platform respectively.
- `faxserver.fail_attempts`: Amount of attempts to send a fax after which it is marked as *failed*.
- `faxserver.fail_retry_secs`: Amount of seconds to wait between "fail\_attempts".
- `faxserver.mail_from`: Sets the e-mail From Header for incoming fax.

### B.1.12 general

The following is the general section:

```
general:
  adminmail: adjust@example.org
  companyname: sipwise
  lang: en
  maintenance: no
  production: yes
  timezone: localtime
```

- `general.adminmail`: Email address used by `monit` to send notifications to.
- `general.companyname`: Label used in `SNMPd` configuration.
- `general.lang`: Sets sounds language (e.g: *de* for German)
- `general.production`: Label to hint self-check scripts about installation mode.
- `general.maintenance`: maintenance mode necessary for safe upgrades.
- `general.timezone`: sip:carrier Timezone

### B.1.13 haproxy

The following is the haproxy section:

```
haproxy:
  admin: 'no'
  admin_port: 8080
  admin_pwd: iKNPFuPFHMCHh9dsXgVg
  enable: 'no'
```

- haproxy.enable: enable haproxy

### B.1.14 heartbeat

The following is the heartbeat section:

```
heartbeat:
  hb_watchdog:
    action_max: 5
    enable: 'yes'
    interval: 10
    transition_max: 10
  pingnodes:
    - 10.60.1.1
    - 192.168.3.4
```

- heartbeat.hb\_watchdog.enable: Enable heartbeat watchdog in order to prevent and fix split brain scenario.
- heartbeat.hb\_watchdog.action\_max: Max errors before taking any action.
- heartbeat.hb\_watchdog.interval: Interval in secs for the check.
- heartbeat.hb\_watchdog.transition\_max: Max checks in transition state.
- heartbeat.pingnodes: List of pingnodes for heartbeat. Minimum 2 entries, otherwise by default NGCP will set the default gateway and DNS servers as pingnodes.

### B.1.15 intercept

The following is the legal intercept section:

```
intercept:
  captagent:
    port: 18090
    schema: http
  enabled: 'no'
```

- intercept.captagent.enable: Enable captagent for Lawful Interception (additional NGCP module).

### B.1.16 kamailio

The following is the kamailio section:

```
kamailio:
  lb:
    cfgt: no
    debug:
      enable: no
      modules:
        - level: '1'
          name: core
        - level: '3'
          name: xlog
    debug_level: '1'
    external_sbc: []
    extra_sockets: ~
    max_forwards: '70'
    mem_log: '1'
    mem_summary: '12'
    nattest_exception_ips:
      - 1.2.3.4
      - 5.6.7.8
    pkg_mem: '16'
    port: '5060'
    remove_isup_body_from_replies: no
    security:
      dos_ban_enable: yes
      dos_ban_time: '300'
      dos_reqs_density_per_unit: '50'
      dos_sampling_time_unit: '5'
      dos_whitelisted_ips: []
      dos_whitelisted_subnets: []
      failed_auth_attempts: '3'
      failed_auth_ban_enable: yes
      failed_auth_ban_time: '3600'
    topoh:
      enable: no
      mask_callid: no
      mask_ip: 127.0.0.8
    shm_mem: '64'
    skip_contact_alias_for_ua_when_tcp:
      enable: no
      user_agent_patterns: []
    start: yes
    strict_routing_safe: no
    syslog_options: yes
    tcp_children: 1
```

```
tcp_max_connections: '2048'
tls:
  enable: no
  port: '5061'
  sslcertfile: /etc/ngcp-config/ssl/myserver.crt
  sslcertkeyfile: /etc/ngcp-config/ssl/myserver.key
udp_children: 1
use_dns_cache: on
proxy:
  allow_info_method: no
  allow_msg_method: no
  allow_peer_relay: no
  allow_refer_method: no
  always_anonymize_from_user: no
  authenticate_bye: no
  cf_depth_limit: '10'
  cfgt: no
  check_prev_forwarder_as_upn: no
  children: 1
  debug:
    enable: no
    modules:
      - level: '1'
        name: core
      - level: '3'
        name: xlog
  debug_level: '1'
  default_expires: '3600'
  default_expires_range: '30'
  dlg_timeout: '43200'
  early_rejects:
    block_admin:
      announce_code: '403'
      announce_reason: Blocked by Admin
    block_callee:
      announce_code: '403'
      announce_reason: Blocked by Callee
    block_caller:
      announce_code: '403'
      announce_reason: Blocked by Caller
    block_contract:
      announce_code: '403'
      announce_reason: Blocked by Contract
    block_in:
      announce_code: '403'
      announce_reason: Block in
    block_out:
      announce_code: '403'
```

```
    announce_reason: Blocked out
block_override_pin_wrong:
    announce_code: '403'
    announce_reason: Incorrect Override PIN
callee_busy:
    announce_code: '486'
    announce_reason: Busy Here
callee_offline:
    announce_code: '480'
    announce_reason: Offline
callee_tmp_unavailable:
    announce_code: '480'
    announce_reason: Temporarily Unavailable
callee_tmp_unavailable_gp:
    announce_code: '480'
    announce_reason: Unavailable
callee_tmp_unavailable_tm:
    announce_code: '408'
    announce_reason: Request Timeout
callee_unknown:
    announce_code: '404'
    announce_reason: Not Found
cf_loop:
    announce_code: '480'
    announce_reason: Unavailable
emergency_invalid:
    announce_code: '404'
    announce_reason: Emergency code not available in this region
emergency_unsupported:
    announce_code: '403'
    announce_reason: Emergency Calls Not Supported
invalid_speeddial:
    announce_code: '484'
    announce_reason: Speed-Dial slot empty
locked_in:
    announce_code: '403'
    announce_reason: Callee locked
locked_out:
    announce_code: '403'
    announce_reason: Caller locked
max_calls_in:
    announce_code: '486'
    announce_reason: Busy
max_calls_out:
    announce_code: '403'
    announce_reason: Maximum parallel calls exceeded
no_credit:
    announce_code: '402'
```

```
    announce_reason: Insufficient Credit
peering_unavailable:
    announce_code: '503'
    announce_reason: PSTN Termination Currently Unavailable
reject_vsc:
    announce_code: '403'
    announce_reason: VSC Forbidden
relaying_denied:
    announce_code: '403'
    announce_reason: Relaying Denied
unauth_caller_ip:
    announce_code: '403'
    announce_reason: Unauthorized IP detected
emergency_priorization:
    enabled: no
    register_fake_200: yes
    register_fake_expires: '3600'
    reject_code: '503'
    reject_reason: Temporary Unavailable
    retry_after: '3600'
enum_suffix: e164.arpa.
expires_range: '30'
filter_100rel_from_supported: no
filter_failover_response: 408|500|503
foreign_domain_via_peer: no
fritzbox:
    enable: no
    prefixes:
    - 0$avp(caller_ac)
    - $avp(caller_cc)$avp(caller_ac)
    - \+$avp(caller_cc)$avp(caller_ac)
    - 00$avp(caller_cc)$avp(caller_ac)
    special_numbers:
    - '112'
    - '110'
    - 118[0-9]{2}
ignore_auth_realm: no
ignore_subscriber_allowed_clis: no
keep_original_to: no
latency_limit_action: '100'
latency_limit_db: '500'
latency_log_level: '1'
latency_runtime_action: 1000
lnp:
    api:
        add_caller_cc_to_lnp_dst: no
        invalid_lnp_routing_codes:
        - ^EE00
```



```
- ^DD00
keepalive_interval: '3'
lnp_request_blacklist: []
lnp_request_whitelist: []
port: '8991'
reply_error_on_lnp_failure: no
request_timeout: '1000'
server: localhost
enabled: no
skip_callee_lnp_lookup_from_any_peer: no
type: api
lookup_peer_destination_domain_for_pbx: no
loop_detection:
  enable: no
  expire: '1'
  max: '5'
max_expires: '43200'
max_gw_lcr: '128'
max_registrations_per_subscriber: '5'
mem_log: '1'
mem_summary: '12'
min_expires: '60'
nathelper:
  sipping_from: sip:pinger@sipwise.local
nathelper_dbro: no
natping_interval: '30'
natping_processes: 1
nonce_expire: '300'
pbx:
  hunt_display_fallback_format: '[H %s]'
  hunt_display_fallback_indicator: $var(cloud_pbx_hg_ext)
  hunt_display_format: '[H %s]'
  hunt_display_indicator: $var(cloud_pbx_hg_displayname)
  hunt_display_maxlength: 8
  ignore_cf_when_hunting: no
peer_probe:
  available_treshold: '1'
  enable: yes
  from_uri_domain: probe.ngcp.local
  from_uri_user: ping
  interval: '10'
  method: OPTIONS
  reply_codes: class=2;class=3;code=403;code=404;code=405
  timeout: '5'
  unavailable_treshold: '1'
perform_peer_failover_on_tm_timeout: yes
perform_peer_lcr: no
pkg_mem: '32'
```

```
port: '5062'
presence:
  enable: yes
  max_expires: '3600'
  reginfo_domain: example.org
proxy_lookup: no
push:
  apns_alert: New call
  apns_sound: incoming_call.xaf
report_mos: yes
set_ruri_to_peer_auth_realm: no
shm_mem: '125'
start: yes
store_recentcalls: no
syslog_options: yes
tcp_children: 1
tm:
  fr_inv_timer: '180000'
  fr_timer: '9000'
treat_600_as_busy: yes
use_enum: no
usrloc_dbmode: '1'
voicebox_first_caller_cli: yes
```

- `kamailio.lb.cfgt`: Enable/disable unit test config file execution tracing.
- `kamailio.lb.debug.enable`: Enable per-module debug options.
- `kamailio.lb.debug.modules`: List of modules to be traced with respective debug level.
- `kamailio.lb.debug_level`: Default debug level for `kamailio-lb`.
- `kamailio.lb.external_sbc`: SIP URI of external SBC used in the Via Route option of peering server.
- `kamailio.lb.extra_sockets`: Add here extra sockets for Load Balancer.
- `kamailio.lb.max_forwards`: Set the value for the Max Forwards SIP header for outgoing messages.
- `kamailio.lb.mem_log`: Specifies on which log level the memory statistics will be logged.
- `kamailio.lb.mem_summary`: Parameter to control printing of memory debugging information on exit or SIGUSR1 to log.
- `kamailio.lb.natatest_exception_ips`: List of IPs that don't need the NAT test.
- `kamailio.lb.shm_mem`: Shared memory used by Kamailio Load Balancer.
- `kamailio.lb.pkg_mem`: PKG memory used by Kamailio Load Balancer.
- `kamailio.lb.port`: Default listen port.
- `kamailio.lb.remove_isup_body_from_replies`: Enable/disable stripping of ISUP part from the message body.

- `kamailio.lb.security.dos_ban_enable`: Enable/Disable DoS Ban.
- `kamailio.lb.security.dos_ban_time`: Sets the ban time.
- `kamailio.lb.security.dos_reqs_density_per_unit`: Sets the requests density per unit (if we receive more than \* `lb.dos_reqs_density_per_unit` within `dos_sampling_time_unit` the user will be banned).
- `kamailio.lb.security.dos_sampling_time_unit`: Sets the DoS unit time.
- `kamailio.lb.security.dos_whitelisted_ips`: Write here the whitelisted IPs.
- `kamailio.lb.security.dos_whitelisted_subnets`: Write here the whitelisted IP subnets.
- `kamailio.lb.security.failed_auth_attempts`: Sets how many authentication attempts allowed before ban.
- `kamailio.lb.security.failed_auth_ban_enable`: Enable/Disable authentication ban.
- `kamailio.lb.security.failed_auth_ban_time`: Sets how long a user/IP has be banned.
- `kamailio.lb.topoh.enable`: Enable topology hiding module (see the [Topology Hiding](#) Section 14.6 subchapter for a detailed description).
- `kamailio.lb.topoh.mask_callid`: if set to `yes`, the SIP Call-ID header will also be encoded.
- `kamailio.lb.topoh.mask_ip`: an IP address that will be used to create valid SIP URIs, after encoding the real/original header content.
- `kamailio.lb.start`: Enable/disable kamailio-lb service.
- `kamailio.lb.strict_routing_safe`: Enable strict routing handle feature.
- `kamailio.lb.syslog_options`: Enable/disable logging of SIP OPTIONS messages to `kamailio-options-lb.log`.
- `kamailio.lb.tcp_children`: Number of TCP worker processes.
- `kamailio.lb.tcp_max_connections`: Maximum number of open TCP connections.
- `kamailio.lb.tls.enable`: Enable TLS socket.
- `kamailio.lb.tls.port`: Set TLS listening port.
- `kamailio.lb.tls.sslcertificate`: Path for the SSL certificate.
- `kamailio.lb.tls.sslcertkeyfile`: Path for the SSL key file.
- `kamailio.lb.udp_children`: Number of UDP worker processes.
- `kamailio.lb.use_dns_cache`: Enable/disable use of internal DNS cache.
- `kamailio.proxy.allow_info_method`: Allow INFO method.
- `kamailio.proxy.allow_msg_method`: Allow MESSAGE method.
- `kamailio.proxy.allow_peer_relay`: Allow peer relay. Call coming from a peer that doesn't match a local subscriber will try to go out again, matching the peering rules.
- `kamailio.proxy.allow_refer_method`: Allow REFER method. Enable it with caution.

- `kamailio.proxy.always_anonymize_from_user`: Enable anonymization of full From URI (as opposed to just From Display-name part by default), has same effect as enabling the preference `anonymize_from_user` for all peers.
- `kamailio.proxy.authenticate_bye`: Enable BYE authentication.
- `kamailio.proxy.cf_depth_limit`: CF loop detector. How many CF loops are allowed before drop the call.
- `kamailio.proxy.cfgt`: Enable/disable unit test config file execution tracing.
- `kamailio.proxy.check_prev_forwarder_as_upn`: Enable/disable validation of the forwarder's number taken from the `Diversion` or `History-Info` header.
- `kamailio.proxy.children`: Number of UDP worker processes.
- `kamailio.proxy.debug.enable`: Enable per-module debug options.
- `kamailio.proxy.debug.modules`: List of modules to be traced with respective debug level.
- `kamailio.proxy.debug_level`: Default debug level for `kamailio-proxy`.
- `kamailio.proxy.default_expires`: Default expires value in seconds for a new registration (for REGISTER messages that contains neither Expires HFs nor expires contact parameters).
- `kamailio.proxy.default_expires_range`: This parameter specifies that the expiry used for the registration should be randomly chosen within `default_expires_range` seconds of the `default_expires` parameter.
- `kamailio.proxy.dlg_timeout`: Dialog timeout in seconds (by default 43200 sec - 12 hours).
- `kamailio.proxy.early_rejects`: Customize here the response codes and sound prompts for various reject scenarios. See the subchapter [Configuring Early Reject Sound Sets](#) Section 5.14.1 for a detailed description.
- `kamailio.proxy.emergency_prioritization.enabled`: Enable an emergency mode support.
- `kamailio.proxy.emergency_prioritization.register_fake_200`: When enabled, generates a fake 200 response to REGISTER from non-prioritized subscriber in emergency mode.
- `kamailio.proxy.emergency_prioritization.register_fake_expires`: Expires value for the fake 200 response to REGISTER.
- `kamailio.proxy.emergency_prioritization.reject_code`: Reject code for the non-emergency request.
- `kamailio.proxy.emergency_prioritization.reject_reason`: Reject reason for the non-emergency request.
- `kamailio.proxy.emergency_prioritization.retry_after`: Retry-After value when rejecting the non-emergency request.

---

**Tip**

In order to learn about details of *emergency prioritization* function of NGCP please refer to Section 5.6 part of the handbook.

---

- `kamailio.proxy.enum_suffix`: Sets ENUM suffix - don't forget . (dot).
- `kamailio.proxy.expires_range`: Set randomization of expires for REGISTER messages (similar to `default_expires_range` but applies to recieved expires value).
- `kamailio.proxy.filter_100rel_from_supported`: Enable filtering of *100rel* from Supported header, to disable PRACK.

- `kamailio.proxy.filter_failover_response`: Response codes with no failover routing required.
- `kamailio.proxy.foreign_domain_via_peer`: Enable/disable of routing of calls to foreign SIP URI via peering servers.
- `kamailio.proxy.fritzbox.enable`: Enable detection for Fritzbox special numbers. Ex. Fritzbox add some prefix to emergency numbers.
- `kamailio.proxy.fritzbox.prefixes`: Fritzbox prefixes to check. Ex. `0$avp(caller_ac)`
- `kamailio.proxy.fritzbox.special_numbers`: Specifies Fritzbox special number patterns. They will be checked with the prefixes defined. Ex. `112`, so the performed check will be `sip:0$avp(caller_ac)112@` if prefix is `0$avp(caller_ac)`
- `kamailio.proxy.ignore_auth_realm`: Ignore SIP authentication realm.
- `kamailio.proxy.ignore_subscriber_allowed_clis`: Set to `yes` to ignore the subscriber's `allowed_clis` preference so that the User-Provided CLI is only checked against customer's `allowed_clis` preference.
- `kamailio.proxy.latency_limit_action`: Limit of runtime in ms for config actions. If a config action executed by `cfg` interpreter takes longer than this value, a message is printed in the logs.
- `kamailio.proxy.latency_limit_db`: Limit of runtime in ms for DB queries. If a DB operation takes longer than this value, a warning is printed in the logs.
- `kamailio.proxy.latency_log_level`: Log level to print the messages related to latency. Default is 1 (INFO).
- `kamailio.proxy.latency_runtime_action`: Limit of runtime in ms for SIP message processing cycle. If the SIP message processing takes longer than this value, a warning is printed in the logs.
- `kamailio.proxy.keep_original_to`: Not used now.
- `kamailio.proxy.lnp.api.add_caller_cc_to_lnp_dst`: Enable/disable adding of caller country code to LNP routing number of the result (`no` by default, LNP result in E.164 format is assumed).
- `kamailio.proxy.lnp.api.invalid_lnp_routing_codes` [only for `api` type]: number matching pattern for routing numbers that represent invalid call destinations; an announcement is played in that case and the call is dropped.
- `kamailio.proxy.lnp.api.keepalive_interval`: Not used now.
- `kamailio.proxy.lnp.api.lnp_request_whitelist` [only for `api` type]: list of matching patterns of called numbers for which LNP lookup must be done.
- `kamailio.proxy.lnp.api.lnp_request_blacklist` [only for `api` type]: list of matching patterns of called numbers for which LNP lookup must not be done.
- `kamailio.proxy.lnp.api.port`: Not used now.
- `kamailio.proxy.lnp.api.reply_error_on_lnp_failure`: Specifies whether platform should drop the call in case of LNP API server failure or continue routing the call to the original callee without LNP.
- `kamailio.proxy.lnp.api.request_timeout` [only for `api` type]: timeout in milliseconds while Proxy waits for the response of an LNP query from *Sipwise LNP daemon*.
- `kamailio.proxy.lnp.api.server`: Not used now.

- `kamailio.proxy.lnp.enabled`: Enable/disable LNP (local number portability) lookup during call setup.
- `kamailio.proxy.lnp.skip_callee_lnp_lookup_from_any_peer`: if set to *yes*, the destination LNP lookup is skipped (has same effect as enabling preference `skip_callee_lnp_lookup_from_any_peer` for all peers).
- `kamailio.proxy.lnp.type`: method of LNP lookup; valid values are: `local` (local LNP database) and `api` (LNP lookup through external gateways). *PLEASE NOTE*: the `api` type of LNP lookup is only available for NGCP PRO / CARRIER installations.
- `kamailio.proxy.lookup_peer_destination_domain_for_pbx`: one of [*yes*, *no*, *peer\_host\_name*] - Sets the content of `destination_domain` CDR field for calls between CloudPBX subscribers. In case of *no* this field contains name of CloudPBX domain; *yes*: peer destination domain; *peer\_host\_name*: human-readable name of the peering server.
- `kamailio.proxy.loop_detection.enable`: Enable the SIP loop detection based on the combination of SIP-URI, To and From header URIs.
- `kamailio.proxy.loop_detection.expire`: Sampling interval in seconds for the incoming INVITE requests (by default 1 sec).
- `kamailio.proxy.loop_detection.max`: Maximum allowed number of SIP requests with the same SIP-URI, To and From header URIs within sampling interval. Requests in excess of this limit will be rejected with 482 Loop Detected response.
- `kamailio.proxy.max_expires`: Sets the maximum expires in seconds for registration.
- `kamailio.proxy.max_gw_lcr`: Defines the maximum number of gateways in `lcr_gw` table
- `kamailio.proxy.max_registrations_per_subscriber`: Sets the maximum registration per subscribers.
- `kamailio.proxy.mem_log`: Specifies on which log level the memory statistics will be logged.
- `kamailio.proxy.mem_summary`: Parameter to control printing of memory debugging information on exit or SIGUSR1 to log.
- `kamailio.proxy.min_expires`: Sets the minimum expires in seconds for registration.
- `kamailio.proxy.nathelper.sipping_from`: Set the From header in OPTIONS NAT ping.
- `kamailio.proxy.nathelper_dbro`: Default is "no". This will be "yes" on CARRIER in order to activate the use of a read-only connection using `LOCAL_URL`
- `kamailio.proxy.natping_interval`: Sets the NAT ping interval in seconds.
- `kamailio.proxy.natping_processes`: Set the number of NAT ping worker processes.
- `kamailio.proxy.nonce_expire`: Nonce expire time in seconds.
- `kamailio.proxy.pbx.hunt_display_fallback_format`: Default is [*H %s*]. Sets the format of the hunt group indicator that is sent as initial part of the From Display Name when subscriber is called as a member of PBX hunt group if the preferred format defined by the `hunt_display_format` and `hunt_display_indicator` can not be used (as in the case of not provisioned subscriber settings). The `%s` part is replaced with the value of the `hunt_display_fallback_indicator` variable.
- `kamailio.proxy.pbx.hunt_display_fallback_indicator`: The internal kamailio variable that sets the number or extension of the hunt group. Default is `$var(cloud_pbx_hg_ext)` which is populated during call routing with the extension of the hunt group.
- `kamailio.proxy.pbx.hunt_display_format`: Default is [*H %s*]. Sets the format of hunt group indicator that is sent as initial part of the From Display Name when subscriber is called as a member of PBX hunt group. This is the preferred (default) indicator format with Display Name, where the `%s` part is replaced with the value of the `hunt_display_indicator` variable.

- `kamailio.proxy.pbx.hunt_display_indicator`: The internal kamailio variable that contains the preferred identifier of the hunt group. Default is `$var(cloud_pbx_hg_displayname)` which is populated during call routing with the provisioned Display Name of the hunt group.
- `kamailio.proxy.pbx.hunt_display_maxlength`: Default is `8`. Sets the maximum length of the variable used as the part of hunt group indicator in Display Name. The characters beyond this limit are truncated in order for hunt group indicator and calling party information to fit on display of most phones.
- `kamailio.proxy.pbx.ignore_cf_when_hunting`: Default is `no`. Whether to disregard all individual call forwards (CFU, CFB, CFT and CFNA) of PBX extensions when they are called via hunt groups. Note that call forwards configured to local services such as Voicebox or Conference are always skipped from group hunting.
- `kamailio.proxy.peer_probe.enable`: Enable the peer probing, must be also checked per individual peer in the panel/API.
- `kamailio.proxy.peer_probe.interval`: Peer probe interval in seconds.
- `kamailio.proxy.peer_probe.timeout`: Peer probe response wait timeout in seconds.
- `kamailio.proxy.peer_probe.reply_codes`: Defines the response codes that are considered successful response to the configured probe request, e.g. `class=2;class=3;code=403;code=404;code=405`, with class defining a code range.
- `kamailio.proxy.peer_probe.unavailable_treshold`: Defines after how many failed probes a peer is considered unavailable.
- `kamailio.proxy.peer_probe.available_treshold`: Defines after how many successful probes a peer is considered available.
- `kamailio.proxy.peer_probe.from_uri_user`: From-userpart for the probe requests.
- `kamailio.proxy.peer_probe.from_uri_domain`: From-hostpart for the probe requests.
- `kamailio.proxy.peer_probe.method`: [OPTIONS|INFO] - Request method for probe request.

---

**Tip**

You can find more information about peer probing configuration in Section [5.10.2](#) of the handbook.

---

- `kamailio.proxy.perform_peer_failover_on_tm_timeout`: Specifies the failover behavior when maximum ring timeout (`fr_inv_timer`) has been reached. In case it is set to `yes`: failover to the next peer if any; in case of `no` stop trying other peers.
- `kamailio.proxy.perform_peer_lcr`: Enable/Disable Least Cost Routing based on peering fees.
- `kamailio.proxy.pkg_mem`: PKG memory used by Kamailio Proxy.
- `kamailio.proxy.shm_mem`: Shared memory used by Kamailio Proxy.
- `kamailio.proxy.port`: SIP listening port.
- `kamailio.proxy.presence.enable`: Enable/disable presence feature
- `kamailio.proxy.presence.max_expires`: Sets the maximum expires value for PUBLISH/SUBSCRIBE message. Defines expiration of the presentity record.
- `kamailio.proxy.presence.reginfo_domain`: Set FQDN of the NGCP domain used in callback for mobile push.
- `kamailio.proxy.push.apns_alert`: Set the content of `alert` field towards APNS.

- `kamailio.proxy.push.apns_sound`: Set the content of `sound` field towards APNS.
- `kamailio.proxy.report_mos`: Enable MOS reporting in the log file.
- `kamailio.proxy.set_ruri_to_peer_auth_realm`: Set R-URI using peer auth realm.
- `kamailio.proxy.start`: Enable/disable kamailio-proxy service.
- `kamailio.proxy.store_recentcalls`: Store recent calls to redis (used by Malicious Call Identification application).
- `kamailio.proxy.syslog_options`: Enable/disable logging of SIP OPTIONS messages to `kamailio-options-proxy.log`.
- `kamailio.proxy.tcp_children`: Number of TCP worker processes.
- `kamailio.proxy.tm.fr_inv_timer`: Set INVITE transaction timeout if no final reply for an INVITE arrives after a provisional message was received (ringing timeout).
- `kamailio.proxy.tm.fr_timer`: Set INVITE transaction timeout if the destination is not responding with provisional response message.
- `kamailio.proxy.treat_600_as_busy`: Enable the 6xx response handling according to RFC3261. When enabled, the 6xx response should stop the serial forking. Also, CFB will be triggered or busy prompt played as in case of 486 Busy response.
- `kamailio.proxy.use_enum`: Enable/Disable ENUM feature.
- `kamailio.proxy.usrloc_dbmode`: Set the mode of database usage for persistent contact storage.
- `kamailio.proxy.voicebox_first_caller_cli`: When enabled the previous forwarder's CLI will be used as caller CLI in case of chained Call Forwards.

### B.1.17 Inpd

The following section defines configuration of LNP daemon, that is used when LNP queries are served by external gateways → the so called LNP API mode.

```
lnpd:
  config:
    daemon:
      foreground: 'false'
      json-rpc:
        ports:
          - '8095'
      loglevel: '6'
      sip:
        port: '5095'
      threads: '4'
    instances:
      default:
        module: sigtran
        destination: 0.0.0.0
        from-domain: voip.example.com
```



```
headers:
  - header: INAP-Service-Key
    value: '2'
reply:
  tcap: raw-tcap
enabled: no
```

- Inpd.enabled: Enable/disable LNP daemon
- Inpd.config: details are shown in [Configuration of LNP daemon](#) Section 5.4.2.3

### B.1.18 mediator

The following is the mediator section:

```
mediator:
  interval: 10
```

- mediator.interval: Running interval of mediator.

### B.1.19 modules

The following is the modules section:

```
modules:
  - enable: no
    name: dummy
    options: numdummies=2
```

- modules: list of configs needed for load kernel modules on boot.
- enable: Enable/disable loading of the specific module (yes/no)
- name: kernel module name
- options: kernel module options if needed

### B.1.20 nginx

The following is the nginx section:

```
nginx:
  status_port: 8081
  xcap_port: 1080
```

- `nginx.status_port`: Status port used by nginx server
- `nginx.xcap_port`: XCAP port used by nginx server

### B.1.21 ntp

The following is the ntp server section:

```
ntp:
  servers:
    - 0.debian.pool.ntp.org
    - 1.debian.pool.ntp.org
    - 2.debian.pool.ntp.org
    - 3.debian.pool.ntp.org
```

- `ntp.servers`: Define your NTP server list.

### B.1.22 ossbss

The following is the ossbss section:

```
ossbss:
  apache:
    port: 2443
    proxyluport: 1080
  restapi:
    sslcertfile: '/etc/ngcp-panel/api_ssl/api_ca.crt'
    sslcertkeyfile: '/etc/ngcp-panel/api_ssl/api_ca.key'
  serveradmin: support@sipwise.com
  servername: "\"myserver\""
  ssl_enable: 'yes'
  sslcertfile: '/etc/ngcp-config/ssl/myserver.crt'
  sslcertkeyfile: '/etc/ngcp-config/ssl/myserver.key'
  frontend: 'no'
  htpasswd:
    -
      pass: '{SHA}w4zj3mxbmynIQ1jsUEjSkN2z2pk='
      user: ngcpsoap
  logging:
    apache:
      acc:
        facility: daemon
        identity: oss
        level: info
      err:
        facility: local7
        level: info
```

```
ossbss:
  facility: local0
  identity: provisioning
  level: DEBUG
web:
  facility: local0
  level: DEBUG
provisioning:
  allow_ip_as_domain: 1
  allow_numeric_usernames: 0
  auto_allow_cli: 1
  carrier:
    account_distribution_function: roundrobin
    prov_distribution_function: roundrobin
  credit_warnings:
    -
      domain: example.com
      recipients:
        - nobody@example.com
      threshold: 1000
  faxpw_min_char: 0
  log_passwords: 0
  no_logline_truncate: 0
  pw_min_char: 6
  routing:
    ac_regex: '[1-9]\d{0,4}'
    cc_regex: '[1-9]\d{0,3}'
    sn_regex: '[1-9]\d+'
  tmpdir: '/tmp'
```

- `ossbss.frontend`: Enable/disable SOAP interface. Set value to `fcgi` to enable old SOAP interface.
- `ossbss.htpasswd`: Sets the username and SHA hashed password for SOAP access. You can generate the password using the following command: `htpasswd -nbs myuser mypassword`.
- `ossbss.provisioning.allow_ip_as_domain`: Allow or not allow IP address as SIP domain (0 is not allowed).
- `ossbss.provisioning.allow_numeric_usernames`: Allow or not allow numeric SIP username (0 is not allowed).
- `ossbss.provisioning.faxpw_min_char`: Minimum number of characters for fax passwords.
- `ossbss.provisioning.pw_min_char`: Minimum number of characters for sip passwords.
- `ossbss.provisioning.log_password`: Enable logging of passwords.
- `ossbss.provisioning.routing`: Regexp for allowed AC (Area Code), CC (Country Code) and SN (Subscriber Number).

### B.1.23 pbx (only with additional cloud PBX module installed)

The following is the PBX section:

```
pbx:
  bindport: 5085
  enable: 'no'
  highport: 55000
  lowport: 50001
  media_processor_threads: 10
  session_processor_threads: 10
  xmlrpcport: 8095
```

- `pbx.enable`: Enable Cloud PBX module.

### B.1.24 prosody

The following is the prosody section:

```
prosody:
  ctrl_port: 5582
  log_level: info
```

- `prosody.ctrl_port`: XMPP server control port.
- `prosody.log_level`: Prosody loglevel.

### B.1.25 pushd

The following is the pushd section:

```
pushd:
  apns:
    enable: yes
    endpoint: api.push.apple.com
    endpoint_port: 0
    extra_instances:
      - certificate: '/etc/ngcp-config/ssl/PushCallkitCert.pem'
        enable: yes
        key: '/etc/ngcp-config/ssl/PushCallkitKey.pem'
        type: callkit
  http2_jwt:
    ec_key: '/etc/ngcp-config/ssl/AuthKey_ABCDE12345.pem'
    ec_key_id: 'ABCDE12345'
    enable: yes
    issuer: 'VWXYZ67890'
```

```
  tls_certificate: ''
  tls_key: ''
  topic: 'com.example.appID'
legacy:
  certificate: '/etc/ngcp-config/ssl/PushChatCert.pem'
  feedback_endpoint: feedback.push.apple.com
  feedback_interval: '3600'
  key: '/etc/ngcp-config/ssl/PushChatKey.pem'
socket_timeout: 0
domains:
- apns:
  endpoint: api.push.apple.com
  extra_instances:
  - certificate: '/etc/ngcp-config/ssl/PushCallkitCert-example.com.pem'
    enable: no
    key: '/etc/ngcp-config/ssl/PushCallkitKey-example.com.pem'
    type: callkit
  http2_jwt:
    ec_key: '/etc/ngcp-config/ssl/AuthKey_54321EDCBA.pem'
    ec_key_id: '54321EDCBA'
    issuer: '09876ZYXWV'
    tls_certificate: ''
    tls_key: ''
    topic: 'com.example.otherAppID'
  legacy:
    certificate: '/etc/ngcp-config/ssl/PushChatCert-example.com.pem'
    feedback_endpoint: feedback.push.apple.com
    key: '/etc/ngcp-config/ssl/PushChatKey-example.com.pem'
  domain: example.com
  enable: yes
  gcm:
    key: 'google_api_key_for_example.com_here'
  enable: yes
  gcm:
    enable: yes
    key: 'google_api_key_here'
  priority:
    call: high
    groupchat: normal
    invite: normal
    message: normal
  muc:
    exclude: []
    force_persistent: 'true'
    owner_on_join: 'true'
  one_device_per_subscriber: no
  port: 45060
  processes: 4
```

```
ssl: yes
sslcertfile: /etc/ngcp-config/ssl/CAsigned.crt
sslcertkeyfile: /etc/ngcp-config/ssl/CAsigned.key
unique_device_ids: no
```

- `pushd.enable`: Enable/Disable the Push Notification feature.
- `pushd.apns.enable`: Enable/Disable Apple push notification.
- `pushd.apns.endpoint`: API endpoint hostname or address. Should be one of *api.push.apple.com* or *api.development.push.apple.com* for the newer HTTP2/JWT based protocol, or one of *gateway.push.apple.com* or *gateway.sandbox.push.apple.com* for the legacy protocol.
- `pushd.apns.endpoint_port`: API endpoint port. Normally 443 or alternatively 2197 for the newer HTTP2/JWT based protocol, or 2195 for the legacy protocol.
- `pushd.apns.legacy`: Contains all options specific to the legacy APNS protocol. Ignored when HTTP2/JWT is in use.
- `pushd.apns.legacy.certificate`: Specify the Apple certificate for push notification https requests from the NGCP to an endpoint.
- `pushd.apns.legacy.key`: Specify the Apple key for push notification https requests from the NGCP to an endpoint.
- `pushd.apns.legacy.feedback_endpoint`: Hostname or address of the APNS feedback service. Normally one of *feedback.push.apple.com* or *feedback.sandbox.push.apple.com*.
- `pushd.apns.legacy.feedback_interval`: How often to poll the feedback service, in seconds.
- `pushd.apns.extra_instances`: If the iOS app supports Callkit push notifications, they can be enabled here and the required separate certificate and key can be specified. Ignored if HTTP2/JWT is enabled.
- `pushd.http2_jwt`: Contains all options specific to the newer HTTP2/JWT based APNS API protocol.
- `pushd.http2_jwt.ec_key`: Name of file that contains the elliptic-curve (EC) cryptographic key provided by Apple, in PEM format.
- `pushd.http2_jwt.ec_key_id`: 10-digit identification string of the EC key in use.
- `pushd.http2_jwt.enable`: Master switch for the HTTP2/JWT based protocol. Disables the legacy protocol when enabled.
- `pushd.http2_jwt.issuer`: Issuer string for the JWT token. Normally the 10-digit team ID string for which the EC key was issued.
- `pushd.http2_jwt.tls_certificate`: Optional client certificate to use for the TLS connection.
- `pushd.http2_jwt.tls_key`: Optional private key for the client certificate to use for the TLS connection.
- `pushd.http2_jwt.topic`: Topic string for the JWT token. Normally the bundle ID for the iOS app.
- `pushd.gcm.enable`: Enable/Disable Google push notification.
- `pushd.gcm.key`: Specify the Google key for push notification https requests from the NGCP to an endpoint.
- `pushd.domains`: Supports a separate set of push configurations (API keys, certificates, etc) for all subscribers of the given domain.
- `pushd.muc.exclude`: list of MUC room jids excluded from sending push notifications.

- `pushd.muc.force_persistent`: Enable/Disable MUC rooms to be persistent. Needed for NGCP app to work with other clients.
- `pushd.muc.owner_on_join`: Enable/Disable all MUC participants to be owners of the MUC room. Needed for NGCP app to work with other clients.
- `pushd.ssl`: The security protocol the NGCP uses for https requests from the app in the push notification process.
- `pushd.sslcertfile`: The trusted certificate file purchased from a CA
- `pushd.sslcertkeyfile`: The key file that purchased from a CA
- `pushd.unique_device_ids`: Allows a subscriber to register the app and have the push notification enabled on more than one mobile device.

### B.1.26 qos

The following is the QOS section:

```
qos:
  tos_rtp: 184
  tos_sip: 184
```

- `qos.tos_rtp`: TOS value for RTP traffic.
- `qos.tos_sip`: TOS value for SIP traffic.

### B.1.27 rate-o-mat

The following is the rate-o-mat section:

```
rateomat:
  enable: 'yes'
  loopinterval: 10
  splitpeakparts: 0
```

- `rateomat.enable`: Enable/Disable Rate-o-mat
- `rateomat.loopinterval`: How long we shall sleep before looking for unrated CDRs again.
- `rateomat.splitpeakparts`: Whether we should split CDRs on peakttime borders.

### B.1.28 redis

The following is the redis section:

```
redis:
  database_amount: 16
  port: 6379
  syslog_ident: redis
```

- `redis.database_amout`: Set the number of databases in redis. The default database is DB 0.
- `redis.port`: Accept connections on the specified port, default is 6379
- `redis.syslog_ident`: Specify the syslog identity.

### B.1.29 reminder

The following is the reminder section:

```
reminder:
  retries: 2
  retry_time: 60
  sip_fromdomain: voicebox.sipwise.local
  sip_fromuser: reminder
  wait_time: 30
  weekdays: '2, 3, 4, 5, 6, 7'
```

- `reminder.retries`: How many times the reminder feature have to try to call you.
- `reminder.retry_time`: Seconds between retries.
- `reminder.wait_time`: Seconds to wait for an answer.

### B.1.30 rsyslog

The following is the rsyslog section:

```
rsyslog:
  elasticsearch:
    action:
      resumeretrycount: '-1'
      bulkmode: 'on'
      dynSearchIndex: 'on'
      enable: 'yes'
      queue:
        dequeuebatchsize: 300
        size: 5000
        type: linkedlist
  external_address:
  external_log: 0
  external_loglevel: warning
  external_port: 514
  external_proto: udp
  ngcp_logs_preserve_days: 93
```

- `rsyslog.elasticsearch.enable`: Enable/Disable Elasticsearch web interface



- `rsyslog.external_address`: Set the remote rsyslog server.
- `rsyslog.ngcp_logs_preserve_days`: Specify how many days to preserve old rotated log files in `/var/log/ngcp/old` path.

### B.1.31 rtpproxy

The following is the rtp proxy section:

```
rtpproxy:
  allow_userspace_only: yes
  cdr_logging_facility: ''
  control_tos: 0
  delete_delay: 30
  dtls_passive: no
  enabled: yes
  final_timeout: 0
  firewall_iptables_chain: ''
  graphite:
    interval: 600
    prefix: rtpengine.
    server: ''
  log_level: '6'
  maxport: '40000'
  minport: '30000'
  num_threads: 0
  prefer_bind_on_internal: no
  recording:
    enabled: no
    mp3_bitrate: '48000'
    nfs_host: 192.168.1.1
    nfs_remote_path: /var/recordings
    output_dir: /var/lib/rtpengine-recording
    output_format: wav
    output_mixed: yes
    output_single: yes
    resample: no
    resample_to: '16000'
    spool_dir: /var/spool/rtpengine
  rtcp_logging_facility: ''
  rtp_timeout: '60'
  rtp_timeout_onhold: '3600'
```

- `rtpproxy.allow_userspace_only`: Enable/Disable the user space failover for rtpengine (*yes* means enable). By default rtpengine works in kernel space.
- `rtpproxy.cdr_logging_facility`: If set, rtpengine will produce a CDR-like syslog line after each call finishes. Must be set to a valid syslog facility string (such as *daemon* or *local0*).

- `rtpproxy.control_tos`: If set to something other than 0, the port used for the control messages is configured to use the given TOS.
- `rtpproxy.delete_delay`: After a call finishes, `rtpproxy` will wait this many seconds before cleaning up resources. Useful for possible late branched calls.
- `rtpproxy.dtls_passive`: If enabled, `rtpproxy` will always advertise itself as a passive role in DTLS setup. Useful in WebRTC scenarios if used behind NAT.
- `rtpproxy.final_timeout`: If set, any calls lasting longer than this many seconds will be terminated, no matter the circumstances.
- `rtpproxy.firewall_iptables_chain`: If set, `rtpproxy` will create an iptables rule for each individual media port opened in this chain.
- `rtpproxy.graphite.interval`: Interval in seconds between sending updates to the Graphite server.
- `rtpproxy.graphite.prefix`: Graphite keys will be prefixed with this string. Must include a separator character (such as a trailing dot) if one should be used.
- `rtpproxy.graphite.server`: Graphite server to send periodic statistics updates to. Disabled if set to an empty string. Must be in format `IP:port` or `hostname:port`.
- `rtpproxy.log_level`: Verbosity of log messages. The default 6 logs everything except debug messages. Increase to 7 to log everything, or decrease to make logging more quiet.
- `rtpproxy.maxport`: Maximum port used by `rtpproxy` for RTP traffic.
- `rtpproxy.minport`: Minimum port used by `rtpproxy` for RTP traffic.
- `rtpproxy.num_threads`: Number of worker threads to use. If set to 0, the number of CPU cores will be used.
- `rtpproxy.recording.enabled`: Enable support for call recording.
- `rtpproxy.recording.mp3_bitrate`: If saving audio as MP3, bitrate of the output file.
- `rtpproxy.recording.nfs_host`: Mount an NFS share from this host for storage.
- `rtpproxy.recording.nfs_remote_path`: Remote path of the NFS share to mount.
- `rtpproxy.recording.output_dir`: Local mount point for the NFS share.
- `rtpproxy.recording.output_format`: Either `wav` for PCM output or `mp3`.
- `rtpproxy.recording.output_mixed`: Create output audio files with all contributing audio streams mixed together.
- `rtpproxy.recording.output_single`: Create separate audio files for each contributing audio stream.
- `rtpproxy.recording.resample`: Resample all audio to a fixed bitrate (*yes* or *no*).
- `rtpproxy.recording.resample_to`: If resampling is enabled, resample to this sample rate.
- `rtpproxy.recording.spool_dir`: Local directory for temporary metadata file storage.
- `rtpproxy.rtcp_logging_facility`: If set, `rtpproxy` will write the contents of all received RTCP packets to syslog. Must be set to a valid syslog facility string (such as `daemon` or `local0`).
- `rtpproxy.rtp_timeout`: Consider a call dead if no RTP is received for this long (60 seconds).
- `rtpproxy.rtp_timeout_onhold`: Maximum limit in seconds for an onhold (1h).

### B.1.32 security

The following is the security section. Usage of the firewall subsection is described in Section [14.2](#):

```
security:
  firewall:
    enable: no
    logging:
      days_kept: '7'
      enable: yes
      file: /var/log/firewall.log
      tag: NGCPFW
  nat_rules4: ~
  nat_rules6: ~
  policies:
    forward: DROP
    input: DROP
    output: ACCEPT
  rules4: ~
  rules6: ~
```

- `security.firewall.enable`: Enable/disable iptables configuration and rule generation for IPv4 and IPv6 (default: `no`)
- `security.firewall.logging.days_kept`: Number of days logfiles are kept on the system before being deleted (log files are rotated daily, default: `7`)
- `security.firewall.logging.enable`: Enables/disables logging of all packets dropped by the NGCP firewall (default: `yes`)
- `security.firewall.logging.file`: File firewall log messages go to (default: `/var/log/firewall.log`)
- `security.firewall.logging.tag`: String prepended to all log messages (internally `DROP` is added to any tag indicating the action triggering the message, default: `NGCPFW`)
- `security.firewall.nat_rules4`: Optional list of IPv4 firewall rules added to table `nat` using iptables-persistent syntax (default: `undef`)
- `security.firewall.nat_rules6`: Optional list of IPv6 firewall rules added to table `nat` using iptables-persistent syntax (default: `undef`)
- `security.firewall.policies.forward`: Default policy for iptables `FORWARD` chain (default: `DROP`)
- `security.firewall.policies.input`: Default policy for iptables `INPUT` chain (default: `DROP`)
- `security.firewall.policies.output`: Default policy for iptables `OUTPUT` chain (default: `ACCEPT`)
- `security.firewall.rules4`: Optional list of IPv4 firewall rules added to table `filter` using iptables-persistent syntax (default: `undef`)
- `security.firewall.rules6`: Optional list of IPv6 firewall rules added to table `filter` using iptables-persistent syntax (default: `undef`)

### B.1.33 sems

The following is the SEMS section:

```
sems:
  bindport: 5080
  conference:
    enable: 'yes'
    max_participants: 10
  debug: 'no'
  highport: 50000
  lowport: 40001
  media_processor_threads: 10
  prepaid:
    enable: 'yes'
  sbc:
    calltimer_enable: 'yes'
    calltimer_max: 3600
    outbound_timeout: 6000
    sdp_filter:
      codecs: PCMA,PCMU,telephone-event
      enable: 'yes'
      mode: whitelist
    session_timer:
      enable: 'yes'
      max_timer: 7200
      min_timer: 90
      session_expires: 300
  session_processor_threads: 10
  vsc:
    block_override_code: 80
    cfb_code: 90
    cfna_code: 93
    cft_code: 92
    cfu_code: 72
    clir_code: 31
    directed_pickup_code: 99
    enable: 'yes'
    park_code: 97
    reminder_code: 55
    speedial_code: 50
    unpark_code: 98
    voicemail_number: 2000
  xmlrpcport: 8090
```

- `sems.conference.enable`: Enable/Disable conference feature.
- `sems.conference.max_participants`: Sets the number of concurrent participant.

- `sems.highport`: Maximum ports used by `sems` for RTP traffic.
- `sems.debug`: Enable/Disable debug mode.
- `sems.lowport`: Minimum ports used by `sems` for RTP traffic.
- `sems.prepaid.enable`: Enable/Disable prepaid feature.
- `sems.sbc.calltimer_max`: Set the default maximum call duration (used if otherwise is not defined by preference).
- `sems.sbc.outbound_timeout`: Set INVITE transaction timeout if the destination is not responding with provisional response message.
- `sems.sbc.session_timer.enable`: If set to "no" all session timer headers are stripped off without considering the session timer related configuration done via the web interface. If set to "yes" the system uses the subscriber/peer configurations values set on the web interface. If set to "transparent" no validation is performed on Session Timer headers, they are ignored by SEMS and therefore negotiated end-to-end.
- `sems.vsc.*`: Define here the VSC codes.

### B.1.34 sms

This section provides configuration of **Short Message Service** on the NGCP. Description of the SMS module is provided earlier in this handbook [here](#) Section 5.27.

In the below example you can see the default values of the configuration parameters.

```
sms:
  core:
    admin_port: '13000'
    smsbox_port: '13001'
  enable: no
  loglevel: '0'
  sendsms:
    max_parts_per_message: '5'
    port: '13002'
  smsc:
    dest_addr_npi: '1'
    dest_addr_ton: '1'
    enquire_link_interval: '58'
    host: 1.2.3.4
    id: default_smsc
    max_pending_submits: '10'
    no_dlr: yes
    password: password
    port: '2775'
    source_addr_npi: '1'
    source_addr_ton: '1'
    system_type: ''
    throughput: '5'
```

```
transceiver_mode: '1'
username: username
```

- sms.core.admin\_port: Port number of admin interface of SMS core module (running on LB nodes).
- sms.core.smsbox\_port: Port number used for internal communication between *bearerbox* module on LB nodes and *smsbox* module on PRX nodes. This is a listening port of the *bearerbox* module (running on LB nodes).
- sms.enable: Set to *yes* if you want to enable SMS module.
- sms.loglevel: Log level of SMS module; the default *0* will result in writing only the most important information into the log file.
- sms.sendsms.max\_parts\_per\_message: If the SM needs to be sent as concatenated SM, this parameter sets the max. number of parts for a single (logical) message.
- sms.sendsms.port: Port number of *smsbox* module (running on PRX nodes).
- sms.smsc. : Parameters of the connection to an SMSC
  - dest\_addr\_npi: Telephony numbering plan indicator for the SM destination, as defined by standards (e.g. *1* stands for E.164)
  - dest\_addr\_ton: Type of number for the SM destination, as defined by standards (e.g. *1* stands for "international" format)
  - enquire\_link\_interval: Interval of SMSC link status check in seconds
  - host: IP address of the SMSC
  - id: An arbitrary string for identification of the SMSC; may be used in log files and for routing SMs.
  - max\_pending\_submits: The maximum number of outstanding (i.e. not acknowledged) SMPP operations between the NGCP and SMSC. As a guideline it is recommended that no more than 10 (default) SMPP messages are outstanding at any time.
  - no\_dlr: Do not request delivery report; when sending an SM and this parameter is set to *yes*, NGCP will not request DR for the message(s). May be required for some particular SMSCs, in order to avoid "Incorrect status report request parameter usage" error messages from the SMSC.
  - password: This is the password used for authentication on the SMSC.
  - port: Port number of the SMSC where NGCP will connect to.
  - source\_addr\_npi: Telephony numbering plan indicator for the SM source, as defined by standards (e.g. *1* stands for E.164)
  - source\_addr\_ton: Type of number for the SM source, as defined by standards (e.g. *1* stands for "international" format)
  - system\_type: Defines the SMSC client category in which NGCP belongs to; defaults to "VMA" (Voice Mail Alert) when no value is given. (No need to set any value)
  - throughput: The max. number of messages per second that NGCP will send towards the SMSC. (Value type: float)
  - transceiver\_mode: If set to *1* (yes / true), NGCP will attempt to use a TRANSCEIVER mode connection to the SMSC. It uses the standard transmit port of the SMSC for receiving SMs too.
  - username: This is the username used for authentication on the SMSC.

### B.1.35 snmpagent

The following is the SNMP Agent section:

```
snmpagent:  
  daemonize: '1'  
  debug: '0'  
  update_interval: '30'
```

- `daemonize`: Enable/Disable `ngcp-snmp-agent` daemonization.
- `debug`: Enable/Disable debug output.
- `update_interval`: Sets the interval in seconds used to update the fetched data.

### B.1.36 sshd

The following is the `sshd` section:

```
sshd:  
  listen_addresses:  
    - 0.0.0.0
```

- `sshd`: specify interface where SSHD should run on. By default `sshd` listens on all IPs found in `network.yml` with type `ssh_ext`. Unfortunately `sshd` can be limited to IPs only and not to interfaces. The current option makes it possible to specify allowed IPs (or all IPs with `0.0.0.0`).

### B.1.37 sudo

The following is in the `sudo` section:

```
sudo:  
  logging: no  
  max_log_sessions: 0
```

- `logging`: enable/disable the I/O logging feature of `sudo`. See man page of `sudo(8)`.
- `max_log_sessions`: when I/O logging is enabled, specifies how many log sessions per individual user `sudo` should keep before it starts overwriting old ones. The default `0` means no limit.

### B.1.38 voisniff

The following is the voice sniffer section:

```
voisniff:
  admin_panel: no
  daemon:
    bpf: 'port 5060 or 5062 or ip6 proto 44 or ip[6:2] & 0x1fff != 0'
    external_interfaces: eth0 eth1
    filter:
      exclude:
        - active: '0'
          case_insensitive: '1'
          pattern: '\ncseq: *\d+ +(register|notify|options)\'
      include: []
    internal_interfaces: lo
  li_x1x2x3:
    call_id:
      suffix:
        - _pbx-1
        - _b2b-1
        - _xfer-1
    client_certificate: ''
    enabled: no
    fix_checksums: no
    fragmented: no
    interface:
      excludes: []
    local_name: sipwise
    x1:
      port: '18090'
  mysql_dump:
    enabled: yes
    num_threads: '4'
  mysql_dump_threads: '4'
  start: no
  threads_per_interface: '10'
  partitions:
    increment: '700000'
    keep: '10'
```

Parameters commonly used for call statistics retrievable on the web interface and for lawful interception:

- `voisniff.daemon.bpf`: Sets the basic packet filter applied by *voisniff-ng* module when capturing packets on network interfaces.
- `voisniff.daemon.external_interfaces`: List of network interfaces where *voisniff-ng* will listen for packets.
- `voisniff.daemon.internal_interfaces`: List of network interfaces that *voisniff-ng* will ignore for capturing packets. These are used for internal communication among NGCP modules. Default: `lo` (→ the loopback interface)
- `voisniff.daemon.filter.exclude` and `voisniff.daemon.filter.include`: Additional filter to determine packets that need to be excluded from / included in capturing.



- `voisniff.daemon.start`: Change to `yes` if you want *voisniff-ng* start at boot. Default is `no`.
- `voisniff.daemon.threads_per_interface`: Controls how many threads per enabled sniffing interface should be launched.

Parameters used only for call statistics:

- `voisniff.admin_panel`: Enable/Disable call statistics on Admin interface. Default: `no`.
- `voisniff.daemon.mysql_dump.*` and `voisniff.daemon.mysql_dump_threads`: These parameters determine how much resource should be dedicated to call statistics collection and storage into the database.
- `voisniff.partitions.*`: These parameters determine how the collected packets are stored in the database: how big chunks are kept in a single table (→ `increment`), how many tables with call data are kept in DB (→ `keep`).

Parameters used only for lawful interception:

- `voisniff.daemon.li_x1x2x3.call_id.suffix`: List of NGCP-internal Call-ID suffix patterns that should be ignored when determining the original SIP Call-ID of an intercepted call.
- `voisniff.daemon.li_x1x2x3.client_certificate`: The client certificate that NGCP uses to connect over TLS to a 3rd party LI provider.
- `voisniff.daemon.li_x1x2x3.enabled`: Set it to `yes` to enable LI services via X1, X2 and X3 interfaces. Default: `no`
- `voisniff.daemon.li_x1x2x3.fix_checksums`: When enabled (= `yes`), NGCP will calculate UDP header checksum for packets sent out on X2 and X3 interfaces. Default: `no`
- `voisniff.daemon.li_x1x2x3.fragmented`: Determines whether *voisniff-ng* is allowed to send fragmented packets via X2 and X3 interfaces. Default: `no`
- `voisniff.daemon.li_x1x2x3.interface.excludes`: This is a list of interfaces that must be excluded from the interception procedures. Default: empty list
- `voisniff.daemon.li_x1x2x3.local_name`: This parameter maps to the `header.source` field of the X2 protocol. It's an arbitrary string and can be used to identify the sending NGCP system. Default: `sipwise`
- `voisniff.daemon.li_x1x2x3.private_key`: The private key that NGCP uses to connect over TLS to a 3rd party LI provider. Only necessary if the client certificate file does not include the private key.
- `voisniff.daemon.li_x1x2x3.x1.port`: The port number on which *voisniff-ng* listens for incoming X1 messages. Default: `18090`

---

**Tip**

Visit Section [16.3.2.3](#) part of the handbook to learn more about lawful interception configuration.

---

### B.1.39 www\_admin

The following is the WEB Admin interface (`www_admin`) section:

```
www_admin:
  ac_dial_prefix: 0
  apache:
    autoprov_port: 1444
  billing_features: 1
  callingcard_features: 0
  callthru_features: 0
  cc_dial_prefix: 00
  conference_features: 1
  contactmail: adjust@example.org
  dashboard:
    enabled: 1
  default_admin_settings:
    call_data: 0
    is_active: 1
    is_master: 0
    read_only: 0
    show_passwords: 1
  domain:
    preference_features: 1
    rewrite_features: 1
    vsc_features: 0
  fastcgi_workers: 2
  fax_features: 1
  fees_csv:
    element_order:
      - source
      - destination
      - direction
      - zone
      - zone_detail
      - onpeak_init_rate
      - onpeak_init_interval
      - onpeak_follow_rate
      - onpeak_follow_interval
      - offpeak_init_rate
      - offpeak_init_interval
      - offpeak_follow_rate
      - offpeak_follow_interval
      - use_free_time
  http_admin:
    autoprov_port: 1444
    port: 1443
    serveradmin: support@sipwise.com
    servername: "\"myserver\""
    ssl_enable: 'yes'
    sslcertfile: '/etc/ngcp-config/ssl/myserver.crt'
```

```
    sslcertkeyfile: '/etc/ngcp-config/ssl/myserver.key'
http_csc:
  autoprov_bootstrap_port: 1445
  autoprov_port: 1444
  port: 443
  serveradmin: support@sipwise.com
  servername: "\"myserver\""
  ssl_enable: 'yes'
  sslcertfile: '/etc/ngcp-config/ssl/myserver.crt'
  sslcertkeyfile: '/etc/ngcp-config/ssl/myserver.key'
logging:
  apache:
    acc:
      facility: daemon
      identity: oss
      level: info
    err:
      facility: local7
      level: info
peer:
  preference_features: 1
peering_features: 1
security:
  password_allow_recovery: 0
  password_max_length: 40
  password_min_length: 6
  password_musthave_digit: 0
  password_musthave_lowercase: 1
  password_musthave_specialchar: 0
  password_musthave_uppercase: 0
  password_sip_autogenerate: 0
  password_sip_expose_subadmin: 1
  password_web_autogenerate: 0
  password_web_expose_subadmin: 1
speed_dial_vsc_presets:
  vsc:
    - '*0'
    - '*1'
    - '*2'
    - '*3'
    - '*4'
    - '*5'
    - '*6'
    - '*7'
    - '*8'
    - '*9'
subscriber:
  auto_allow_cli: 0
```

```
extension_features: 0
voicemail_features: 1
```

- `www_admin.http_admin.*`: Define the Administration interface and certificates.
- `www_admin.http_csc.*`: Define the Customers interface and certificates.
- `www_admin.contactmail`: Email to show in the GUI's Error page.

## B.2 constants.yml Overview

`/etc/ngcp-config/constants.yml` is one of the main configuration files that contains important (static) configuration parameters, like NGCP system-user data.



### Caution

NGCP platform administrator should not change content of `constants.yml` file unless absolutely necessary. Please contact Sipwise Support before changing any of the parameters within the `constants.yml` file!

## B.3 network.yml Overview

`/etc/ngcp-config/network.yml` is one of the main configuration files that contains network-related configuration parameters, like IP addresses and roles of the node(s) in sip:carrier system.

The next example shows a part of the `network.yml` configuration file. Explanation of all the configuration parameters is provided in [Network Configuration](#) Section 10 section of the handbook.

### Sample host configuration for sip:carrier

```
web01a:
  bond0:
    bond_miimon: '100'
    bond_mode: active-backup
    bond_slaves: 'eth0 eth1'
    hwaddr: 00:00:00:00:00:00
    ip: 192.168.1.2
    netmask: 255.255.255.0
    shared_ip:
      - 192.168.1.1
    type:
      - boot_int
  eth0:
    hwaddr: 00:00:00:00:00:00
  eth1:
    hwaddr: 00:00:00:00:00:00
  interfaces:
```

```
- vlan11
- vlan666
- vlan35
- vlan100
- vlan80
- vlan90
- vlan15
- vlan20
- lo
- eth0
- eth1
- bond0
lo:
  advertised_ip: []
  hwaddr: 00:00:00:00:00:00
  ip: 127.0.0.1
  netmask: 255.0.0.0
  shared_ip: []
  shared_v6ip: []
  type:
    - ssh_ext
    - api_int
  v6ip: '::1'
peer: web01b
role:
  - mgmt
status: 'online'
vlan20:
  advertised_ip: []
  hwaddr: 00:00:00:00:00:00
  ip: 172.31.3.75
  netmask: 255.255.255.240
  shared_ip:
    - 172.31.3.74
  type:
    - web_int
  vlan_raw_device: bond0
  post_up:
    - 'route add -host 172.30.172.247 gw 172.31.3.65 dev vlan20'
vlan100:
  hwaddr: 00:0a:f7:8d:32:ec
  ip: 172.31.3.5
  netmask: 255.255.255.224
  shared_ip:
    - 172.31.3.4
  type:
    - ha_int
    - web_int
```

```
- ssh_ext
vlan_raw_device: bond0
vlan11:
  dns_nameservers:
    - 172.31.3.244
    - 192.168.56.11
    - 192.168.57.11
  gateway: 172.31.3.33
  hwaddr: 00:00:00:00:00:00
  ip: 172.31.3.37
  netmask: 255.255.255.224
  shared_ip:
    - 172.31.3.36
  shared_v6ip: []
  type:
    - mon_ext
    - ssh_ext
  vlan_raw_device: bond0
vlan15:
  hwaddr: 00:00:00:00:00:00
  ip: 192.168.181.201
  netmask: 255.255.255.0
  post_up:
    - 'route add -net 172.25.240.0/24 gw 192.168.181.1 dev vlan15'
    - 'route add -net 192.168.6.0/24 gw 192.168.181.1 dev vlan15'
  shared_ip:
    - 192.168.181.200
  type:
    - ssh_ext
    - web_int
    - mon_ext
  vlan_raw_device: bond0
vlan35:
  hwaddr: 00:00:00:00:00:00
  ip: 172.31.3.101
  netmask: 255.255.255.240
  shared_ip:
    - 172.31.3.100
  type:
    - sip_int
  vlan_raw_device: bond0
vlan666:
  hwaddr: 00:00:00:00:00:00
  ip: 46.5.10.37
  netmask: 255.255.255.240
  shared_ip:
    - 46.5.10.36
  type:
```

```
    - web_ext
    vlan_raw_device: bond0
vlan80:
  hwaddr: 00:00:00:00:00:00
  ip: 172.31.3.237
  netmask: 255.255.255.248
  shared_ip:
    - 172.31.3.236
  type:
    - phone_ext
    - web_ext
  vlan_raw_device: bond0
  post_up:
    - 'ip route add default via 172.31.3.233 dev vlan80 table phones_ext'
    - 'ip rule add from 172.31.3.236 lookup phones_ext prio 1000'
vlan90:
  hwaddr: 00:00:00:00:00:00
  ip: 46.5.10.53
  netmask: 255.255.255.248
  post_up:
    - 'route add -host 77.244.249.93 gw 46.5.10.49 dev vlan90'
  shared_ip:
    - 46.5.10.52
  type:
    - repos_ext
  vlan_raw_device: bond0
```

## C NGCP-Faxserver Configuration

For an overview of Faxserver architecture and features, please see the [Faxserver](#) Section 5.11 chapter.

### C.1 Faxserver Components

Starting from mr4.3 release there is a completely reworked fax server in a form of standalone daemon that uses Asterisk as its transmission component. No other component—such as `hylafax` or `iaxmodem`—is necessary to send and receive faxes on sip:carrier platform.

### C.2 Enabling Faxserver

In order to configure functions of NGCP Faxserver one needs to update the main NGCP configuration file `/etc/ngcp-config/config.yml` with the correct fax options:

```
faxserver:
  enable: yes
  fail_attempts: '3'
  fail_retry_secs: '60'
  keep_failed_fax: yes
  keep_failed_fax_days: '60'
  keep_received_fax: yes
  keep_received_fax_days: '60'
  keep_sent_fax: yes
  keep_sent_fax_days: '60'
  mail_from: 'Sipwise NGCP FaxServer <voipfax@ngcp.sipwise.local>'
```

Parameters are:

- `enable`: must be `yes` to enable Faxserver
- `fail_...` : the number and timeout of fax sending retrials
- `keep_...` : fax retention definitions: enabling and length in days
- `mail_from`: the *From* header in the e-mail that is sent by Fax2Mail feature when a fax is received



#### Important

Ensure that in `network.yml` the `api_int` interface is assigned to the appropriate network interface or a VLAN of the node with the `mgmt` role. Usually, this is the same network interface or VLAN where the `ha_int` interface is assigned to. The `api_int` interface must be removed from all other nodes.

---



### C.3 Fax Templates Configuration

One needs to update `/etc/ngcp-config/templates/etc/ngcp-faxserver/faxserver.conf.tt2` if he wants to use custom content in the fax and e-mail templates that are used by Faxserver to generate the actual fax or e-mail. This may be done under the "User templates" section in the file.

#### Applying new Faxserver configuration

Once the above mentioned configuration files have been modified the new settings must be applied:

```
ngcpcfg apply 'Configured fax server'  
ngcpcfg push all
```

### C.4 Fax Services Configuration per Subscriber

Fax services must be explicitly activated for subscribers before they can send or receive faxes. This activation and the custom settings may be set on the NGCP Web panel in the following way (as an administrator):

- Go to *Subscribers* and find the subscriber that you want to modify settings for
- Click on *Preferences* button
- Select *FaxFeatures*

In both sections *Fax2Mail and SendFax* and *Mail2Fax* there is a field: *Active*. This must be changed from *no* to *yes* if the particular fax service must be activated.

When fax services have been activated the user sees a summary of settings in *FaxFeatures* section on his Preferences page:

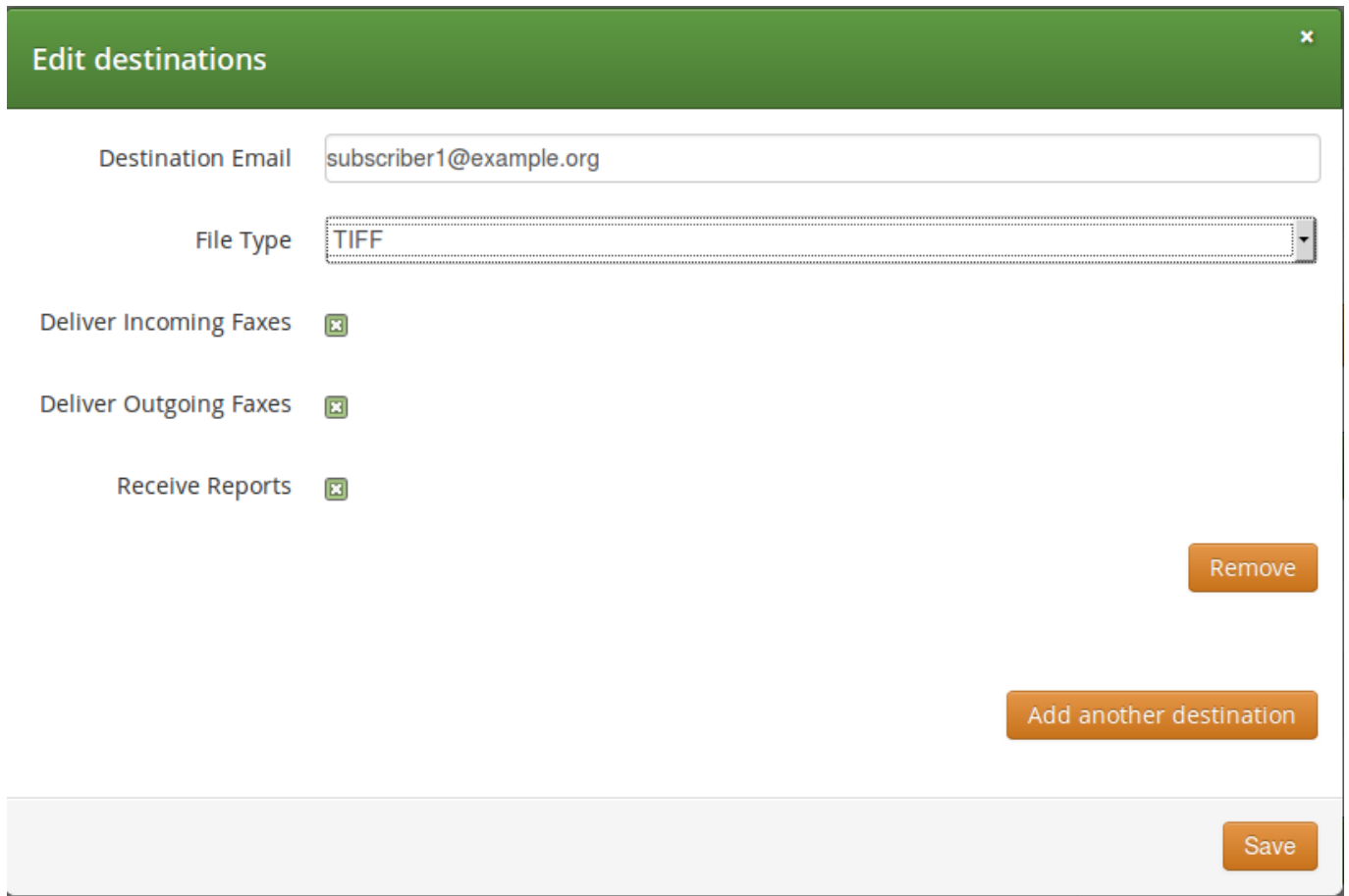
Voicemail and Voicebox		
Fax Features		
Fax2Mail and Sendfax		
Name	Value	
Name in Fax Header for Sendfax		
Active	yes	
Destinations	subscriber1@example.org as TIFF	
Mail2Fax		
Name	Value	
Active	yes	
Secret Key (empty=disabled)		
Secret Key Renew	never	
Last Secret Key Modify Time		
Secret Key Renew Notify		
ACL	regex from_email <u>subscriber1@example.org</u> and received_from <u>any</u> to <u>^4399.+</u> destination	
Speed Dial		

Figure 164: Fax Settings

Details of Fax2Mail, SendFax and Mail2Fax settings are described in subsequent paragraphs.

## C.5 Fax2Mail and SendFax Settings

- **Name in Fax Header for SendFax:** optional field that contains the subscribers name on faxes sent from the Web panel directly
- **Destinations:** e-mail addresses and selections of notification items that define about which event and where an e-mail is sent; this is a list of such definitions



**Edit destinations** ✕

Destination Email

File Type

Deliver Incoming Faxes

Deliver Outgoing Faxes

Receive Reports

Figure 165: Fax2Mail Destination

The parameters for a destination are as follows:

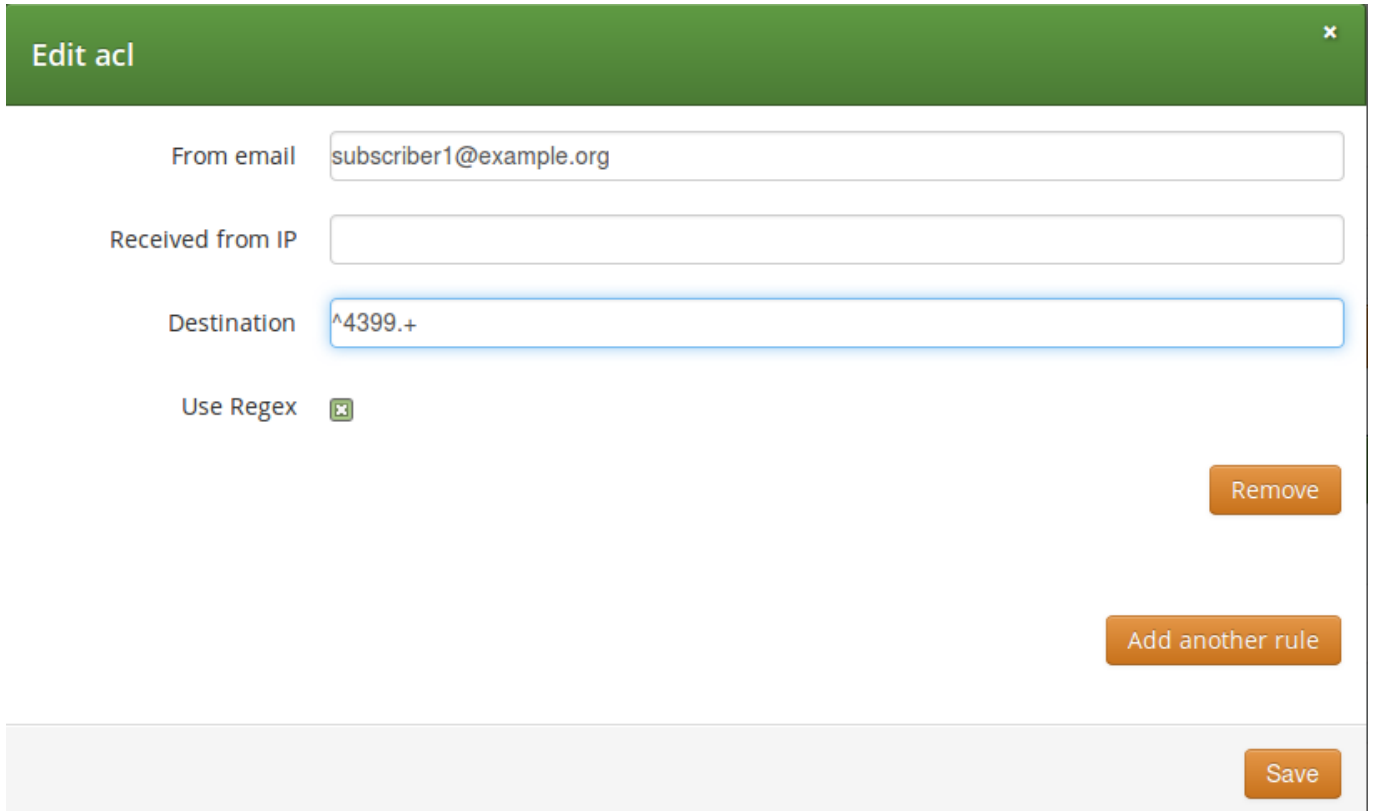
- `Destination Email`: the e-mail address where the notification must be sent
- `File Type`: file format of faxes attached to e-mails
- `Deliver Incoming Faxes`: select this in order to receive incoming faxes in e-mail
- `Deliver Outgoing Faxes`: select this in order to receive a report about sent faxes
- `Receive Reports`: select this in order to receive reports about success / failure of fax transmissions

## C.6 Mail2Fax Settings

A subscriber can restrict access to his Mail2Fax service with some methods, those can also be combined:

- using a *secret key* that is only known to him, and is inserted in every mail that he sends to NGCP to be forwarded as fax
- using an *access control list (ACL)* that determines from which endpoint and for which destination a mail-to-fax is accepted by NGCP platform

- **Secret Key:** the secret key used to validate the sender of an e-mail; not used if left empty
- **Secret Key Renew:** secret key renewal period; NGCP platform will enforce renewal of the secret key when the defined time has elapsed
- **Last Secret Key Modify Time:** information about the last secret key modification time
- **Secret Key Renew Notify:** an e-mail address where the notification about secret key modification is sent
- **ACL:** access control list, see the details below; this is a list of access control rules



The screenshot shows a web interface for editing an Access Control List (ACL) rule. The window title is "Edit acl". It features four input fields: "From email" containing "subscriber1@example.org", "Received from IP" which is empty, "Destination" containing the regular expression "^4399.+", and a "Use Regex" checkbox that is checked. On the right side, there are three buttons: "Remove" (top), "Add another rule" (middle), and "Save" (bottom).

Figure 166: Mail2Fax Access Control List

The parameters for access control rules:

- **From email:** this sender is allowed to use Mail2Fax service
- **Received from IP:** this IP address or host name must be present in From e-mail header
- **Destination:** either a complete phone number in E.164 format, or a regular expression ("Use Regex" checkbox must be ticked) that may define a range of numbers. Examples: "4313334445" as a single number; "^4399.+" as a regular expression: all destinations starting with "4399"

**Caution**

When neither *Secret Key*, nor *ACL* is defined then Mail2Fax service will deny accepting any e-mail for sending faxes!

## C.7 Sending Fax from Web Panel

A subscriber can log in to his *Customer Self Care* website and send faxes directly from there. In order to do this, one needs to do the following:

- Go to *Settings* → *Web Fax* page

**Tip**

The list of received faxes is also available here.

- Press *Send Fax* button to start entering data, such as recipient and content for the fax being sent:

The screenshot shows a web panel titled "Create Fax" with a green header and a close button (X) in the top right corner. The form contains the following fields:

- Destination Number:** A text input field containing "43993003".
- Quality:** A dropdown menu with "Normal" selected.
- Page header:** A text input field containing "Test fax".
- Content:** A large text area containing "This is the plain text content, but there will also be an attachment."
- File:** A file selection area with a "Browse..." button and the filename "handbook-pro.pdf" displayed.

At the bottom right of the form is an orange "Send" button.

Figure 167: Sending Fax from Web Panel

Both plain text message and attached files can be sent in the fax. First page(s) will contain the plain text message and the content of attached files will follow that.

## C.8 Faxserver Mail2Fax Configuration

Using NGCP Faxserver's Mail2Fax service requires the configuration of sip:carrier's local mail server that is *Exim*. It has to be configured in a way that it can receive mails from outside of the server, because *Exim* by default listens only on the local interfaces for incoming mails.

### Exim Configuration

The NGCP platform administrator must reconfigure *Exim* in order to enable receiving e-mails for fax sending:

```
dpkg-reconfigure exim4-config
```

*PLEASE NOTE:* When entering configuration data the following points must be kept in mind:

- operation mode has to be set to "mail sent by smarthost; no local mail"
- "mail2fax.example.org" must be added to accepted domains, where "example.org" is the domain name of the NGCP platform operator

### DNS Configuration

It is necessary to add a subdomain starting as `mail2fax.` to the list of domain names. That is where the faxes will be sent by users to trigger Mail2Fax service.

---

#### Tip

Alternatively, edit `/etc/ngcp-config/templates/etc/exim4/conf.d/router/999_mail2fax.tt2` file and adjust it to your personal preferences. Although this is not recommended and should only be done by Sipwise support engineers.

---

## C.9 Sending Fax Using E-mail Clients

When sending an e-mail that should be converted to a fax, there are some points to keep in mind so that Faxserver properly processes the e-mail.

- **To header:**
  - must contain the subscriber's number who is sending the fax, as the username part of the mail address
  - must contain the specific domain starting with `mail2fax.`
- **Subject header:** must contain the fax destination number
- **Body** should consist of plain text data

- Adding **attachments** is possible, but only plain text and PDF formats are supported

### Secret Key

In order to use the "secret key" access control feature, it should be either put in the first row of the e-mail body followed by an empty line, or included as a plain text attachment. Once it has been validated, it will be removed from the email.



#### Important

Either add the secret key to the body, or attach it. Never do both as only one will be recognized and removed, leaving the other one to be sent as part of the fax.

---

### Mail Example

Provided there is a subscriber on sip:carrier platform with the 43130111 number, the destination fax is 43130222 and the secret key is "MySecretKey":

```
From: User Name <username@example.org>
To: 43130111@mail2fax.example.org
Subject: 43130222
-----
MySecretKey

This is a test fax.

Cheers
```

## C.10 Managing Faxes via the REST API

It is possible to send and receive faxes and configure fax settings using the built-in REST API interface.

In subsequent sections you can find examples of using the API for sending, receiving faxes and changing fax settings.

### C.10.1 Configuring Fax Settings

#### C.10.1.1 Retrieving Fax Settings

The following example retrieves the fax settings for the subscriber with ID 3.

```
Method: GET
Content-Type: application/hal+json

https://127.0.0.1:1443/api/faxserversettings/3
```

The output format is as follows (only the relevant output data is shown):

```
"active" : true,
"destinations" : [
  {
    "destination" : "user@company.com",
    "filetype" : "PDF14",
    "incoming" : true,
    "outgoing" : true,
    "status" : true
  }
],
"name" : null,
"password" : null
```

### C.10.1.2 Updating Fax Settings

The following example updates a specific parameter. Namely, it deactivates the fax feature for the subscriber with ID 3.

```
Method: PATCH
Content-Type: application/json-patch+json

https://127.0.0.1:1443/api/faxserversettings/3

--data-binary '[ { "op" : "replace", "path" : "/active", "value" : 0 } ]'
```

### C.10.2 Sending a Fax

The following request sends a PDF file located at `/tmp/test_fax.pdf` as fax to 431110002 from the subscriber with ID 3.

```
Method: POST
Content-Type: multipart/form-data

https://127.0.0.1:1443/api/faxes/

--form 'json={"destination" : "431110002", "subscriber_id" : 3}' --form 'faxfile=@/tmp/ ↵
test_fax.pdf'
```

### C.10.3 Receiving a Fax

All received faxes are stored on the server and can be retrieved on demand. You can retrieve a stored fax by following these steps:

1. Firstly, obtain the internal ID of the fax:

```
Method: GET
Content-Type: application/json
```



```
https://127.0.0.1:1443/api/faxes/3
```

This request returns the list of stored faxes for the subscriber with ID 3. One of the available faxes is returned like this:

```
"callee" : "431110002",
"caller" : "431110001",
"direction" : "out",
"duration" : "0",
"filename" : "d9799276-b7d9-454f-98c3-714edf7e3072.tif",
"id" : 5,
"pages" : "1",
"quality" : "8031x7700",
"reason" : "Normal Clearing / SIP 200 OK [1/3]",
"signal_rate" : "14400",
"status" : "SUCCESS",
"subscriber_id" : 1,
"time" : "2016-07-30 09:49:59"
```

2. Now, to retrieve the fax with ID 5, use the following request:

```
Method: GET
Content-Type: application/hal+json

https://127.0.0.1:1443/api/faxerecordings/5
```

By default, the fax is in the TIFF format. It is also possible to request it in a different format. To retrieve the same fax in PDF14, use the following request:

```
https://127.0.0.1:1443/api/faxerecordings/5?format=pdf14
```

## C.10.4 Configuring Mail2Fax Settings

The configuration of Mail2Fax settings via the REST API is similar to the fax settings configuration.

### C.10.4.1 Retrieving Mail2Fax Configuration

To get the Mail2Fax configuration for the subscriber with ID 3, use the following request:

```
Method: GET
Content-Type: application/hal+json

https://127.0.0.1:1443/api/mailtofaxsettings/3
```

The output format is as follows (only the relevant output data is shown):

```

"acl" : [],
"active" : false,
"secret_key" : "secretkeypassword",
"secret_key_renew" : "daily",
"secret_renew_notify" : [
  {
    "destination" : "user1@company.com"
  }
]

```

#### C.10.4.2 Updating Mail2Fax Configuration

The following set of requests changes the Mail2Fax configuration with new secret key settings.

- Secret key value:

```

Method: PATCH
Content-Type: application/json-patch+json

https://127.0.0.1:1443/api/faxserversettings/3

--data-binary '[ { "op" : "replace", "path" : "/secret_key", "value" : " ←
  newsecretkeypassword" } ]'

```

- Secret key renewal interval:

```

Method: PATCH
Content-Type: application/json-patch+json

--data-binary '[ { "op" : "replace", "path" : "/secret_key_renew", "value" : "monthly" } ←
  ]'

```

- List of email addresses that receive the automatic secret key update notifications:

```

Method: PATCH
Content-Type: application/json-patch+json

--data-binary '[ { "op" : "replace", "path" : "/secret_renew_notify", "value" : [ { " ←
  destination": "user2@company.com" }, { "destination": "user3@company.com" } ] } ]'

```

#### C.10.5 Using Advanced Faxserver and Mail2Fax Settings via the REST API

On the NGCP REST API documentation web page you can find the complete list of available Faxserver and Mail2Fax configuration parameters: [https://<ngcp\\_ip\\_address>:1443/api](https://<ngcp_ip_address>:1443/api)

**Important**

The information on the web page is relevant for your platform version and may change in next releases.

After visiting the API documentation main page, you can find the following entries related to Faxserver operations:

- Faxes ([https://<ngcp\\_ip\\_address>:1443/api/#faxes](https://<ngcp_ip_address>:1443/api/#faxes))
- FaxRecordings ([https://<ngcp\\_ip\\_address>:1443/api/#faxrecordings](https://<ngcp_ip_address>:1443/api/#faxrecordings))
- FaxserverSettings ([https://<ngcp\\_ip\\_address>:1443/api/#faxserversettings](https://<ngcp_ip_address>:1443/api/#faxserversettings))

## C.11 Troubleshooting

The following log file may be used to check Faxserver functionality: `/var/log/ngcp/faxserver.log`

### C.11.1 Session ID (SID)

Faxserver stores basic information about each processed fax in a session file. The most important element within this set of data is the *Session ID* (SID) that uniquely identifies a fax throughout its lifetime.

Session ID is a long hexadecimal string (a kind of UUID) that can be read from the above mentioned Faxserver logfile, and which itself is used also as the filename in files that belong to a specific sent / received fax. An example:

```
root@sp1:~# cat /var/spool/ngcp/faxserver/failed/1e480167-5de6-4cc2-948b-de58d1a0bb8c.err

created: 2016-09-06 04:41:32
caller: 111111111
callee: 222222222
file: 1e480167-5de6-4cc2-948b-de58d1a0bb8c.tif
sid: 1e480167-5de6-4cc2-948b-de58d1a0bb8c
dir: out
attempts: 0
fail_attempts: 3
fail_retry_secs: 60
quality: normal
status: FAILED
error: Internal error
modified: 2016-09-06 17:41:30

root@sp1:~#
```

The data element `sid` is the session ID. Other important elements are:

- `caller` and `callee`: these are probably searched for when trying to figure out what happened to a specific fax transmission, if you don't know the SID

- `dir`: direction of fax transmission: *in'coming* or *'out'going* or *'mtf* for mail-to-fax
- `status`: shows success or failure
- `error`: the error cause in case of failed faxes

### C.11.2 Fax Storage Location

Faxserver stores all of its processed faxes at the path: `/var/spool/ngcp/faxserver/...` Within that directory the most relevant subdirectories are `failed` and `completed` that store the SID file and the fax itself in TIFF format of those faxes that failed or were successful, respectively.

## D RTC:engine

### D.1 Overview

WebRTC is an open project providing browsers and mobile applications with Real-Time Communications (RTC) capabilities. The RTC:engine protocol is a light weight messaging and signaling protocol for WebSocket clients. Technically it is a WebSocket sub protocol. It consists of JSON messages that are used to initiate and control call dialogs, send chat messages, join and control conferences and share files. It is similar to well known signaling protocols like SIP, but much simpler. It does not care about the underlying network protocols, like SIP does.

### D.2 RTC:engine enabling

The RTC:engine is not activated by default and needs a few steps to setup.

#### D.2.1 Enabling services via CLI

First you have to enable it first on your server via CLI. Connect with SSH on your server, open `/etc/ngcp-config/config.yml` with your editor of choice and change the following properties:

```
fileshare:
  enable: yes

rtcengine:
  conference:
    relay:
      app_id: bormuth
      url: http://xms.sipwise.com:81
  call:
    relay:
      app_id: bormuth
      url: http://xms.sipwise.com:81
  enable: yes
  expose_provisioning_api: yes

www_admin:
  http_csc:
    servername: '$IP_OF_VM'
```

Save the config.yml file and run `$ ngcpcfg apply "enable rtcengine"`. After the script ran, check the status of all services via `$ monit summary` or `$ monit status`.

## D.2.2 Enabling via Panel for resellers and subscribers

The WebRTC subscriber is just a normal subscriber which has just a different configuration in his Preferences. You need to change the following preferences under *Subscribers*→*Details*→*Preferences*→*NAT and Media Flow Control*:

- **use\_rtproxy**: Always with rtproxy as additional ICE candidate
- **transport\_protocol**: RTP/SAVPF (encrypted SRTP with RTCP feedback)

The `transport_protocol` setting may change, depending on your WebRTC client/browser configuration. Supported protocols are the following:

- Transparent (Pass through using the client's transport protocol)
- RTP/AVP (Plain RTP)
- RTP/SAVP (encrypted SRTP)
- RTP/AVPF (RTP with RTCP feedback)
- RTP/SAVPF (encrypted SRTP with RTCP feedback)
- UDP/TLS/RTP/SAVP (Encrypted SRTP using DTLS)
- UDP/TLS/RTP/SAVPF (Encrypted SRTP using DTLS with RTCP feedback)



### Warning

The below configuration is enough to handle a WebRTC client/browser. As mentioned, you may need to tune a little bit your `transport_protocol` configuration, depending on your client/browser settings.

In order to have a bridge between normal SIP clients (using plain RTP for example) and WebRTC client, the normal SIP clients' preferences have to have the following configuration:

**transport\_protocol**: RTP/AVP (Plain RTP)

This will teach Sip Provider to translate between Plain RTP and RTP/SAVPF when you have calls between normal SIP clients and WebRTC clients.

## D.2.3 Create RTC:engine session

### D.2.3.1 Create sessions

#### Request:

```
curl -i -X POST --insecure --user SUBSCRIBER_ID:SUBSCRIBER_PW -H 'Content-Type: application ←
  /json' --data-binary '{} ' https://IP_OF_VM/api/rtcsessions/
```

**Response Header:**

```
Location: /api/rtcsessions/7
```

**D.2.3.2 Receive sessions****Request:**

```
curl -i -X GET --insecure --user SUBSCRIBER_ID:SUBSCRIBER_PW -H 'Content-Type: application/ ↵  
json' https://IP_OF_VM/api/rtcsessions/{ID_FROM_LAST_REQUEST_HEADER}
```

**Response Header:**

```
{  
  ...  
  "rtc_app_name" : "default_default_app",  
  "rtc_browser_token" : "22fz8e51-ad6e-481e-a389-15c58c3fe5ac",  
  "rtc_network_tag" : "",  
  "subscriber_id" : "263"  
}
```

---

**Tip**

Use `rtc_browser_token` in your `cdk.Client`.

---

**D.3 RTC:engine protocol details****D.3.1 Terminology****D.3.1.1 Connector**

There are two kinds of connectors. The front and the back connectors. The only front connector is the `BrowserConnector`. It has access to all `WebSocket` connections and is responsible for delivering RCT:engine protocol messages to the `WebSocket` clients, and for forwarding messages from the `WebSocket` clients to the router.

Currently there are four back connectors (`SipConnector`, `XmppConnector`, `WebrtcConnector`, `ConferenceConnector`). Every back connector implements a certain communication use case.

**D.3.1.2 Router**

The router is very simple stateless message broker, that is responsible for delivering the messages to the right connector. To decide where to send the message, the router takes a look at the recipient address (to) and forwards the message to the specified connector.

### D.3.1.3 User

### D.3.1.4 App

An app is a scope for a certain RTC:engine integration. Every user can have multiple apps. And an app contains sessions.

### D.3.1.5 Network

A network is a user wide configuration, that maps a custom network name (tag) to a certain back connector. Additionally it can also store network specific configurations. And any account that is related to a certain network, will merge its custom configs with the network configs, and send its messages to the specified connector.

### D.3.1.6 Session

### D.3.1.7 Account

An account represents the credentials for a specific network. Usually it consists of an identifier like a SIP uri (sip:user@domain.tld) and an access token or rather a password.

### D.3.1.8 Browser SDK

The Browser SDK is an abstraction layer on top of the RTC:engine protocol. It is served as bundled javascript library, and provides convenient components and methods for all use cases.

## D.3.2 Messages

A typical message created by the browser sdk contains the following fields:

```
{
  "method": "module.action",
  "from": "connector:id",
  "to": "connector:id",
  "session": "session",
  "body": {
    ...
  }
}
```

### D.3.2.1 Fields

### D.3.2.2 method

It is separated in two parts. The first part is the module. It is a delegation key to separate concerns in the code. The second part is the action, which represents a specific method in a module.



### D.3.2.3 from

It represents the current sender of a message. For example the user creates a new call via the browser sdk, the message would look like this:

```
{
  "method": "call.start",
  "from": "",
  "to": "webrtc:b2bua1",
  "session": "session1",
  "body": {
    ...
  }
}
```

The content of the field is completely irrelevant, because the BrowserConnector will overwrite this field. The reason is to avoid user manipulation.

```
{
  "method": "call.start",
  "from": "browser:ws1",
  "to": "webrtc:b2bua1",
  "session": "session1",
  "body": {
    ...
  }
}
```

### D.3.2.4 to

In general this field represents the recipient of a message. The recipients address consists of two parts. First part is the prefix that targets the connector. Second part is the identifier of the recipient.

### D.3.2.5 session

If you provisioned with the RTCEngine, you get a session and its token property. The browser SDK adds this token to every message.

### D.3.2.6 body

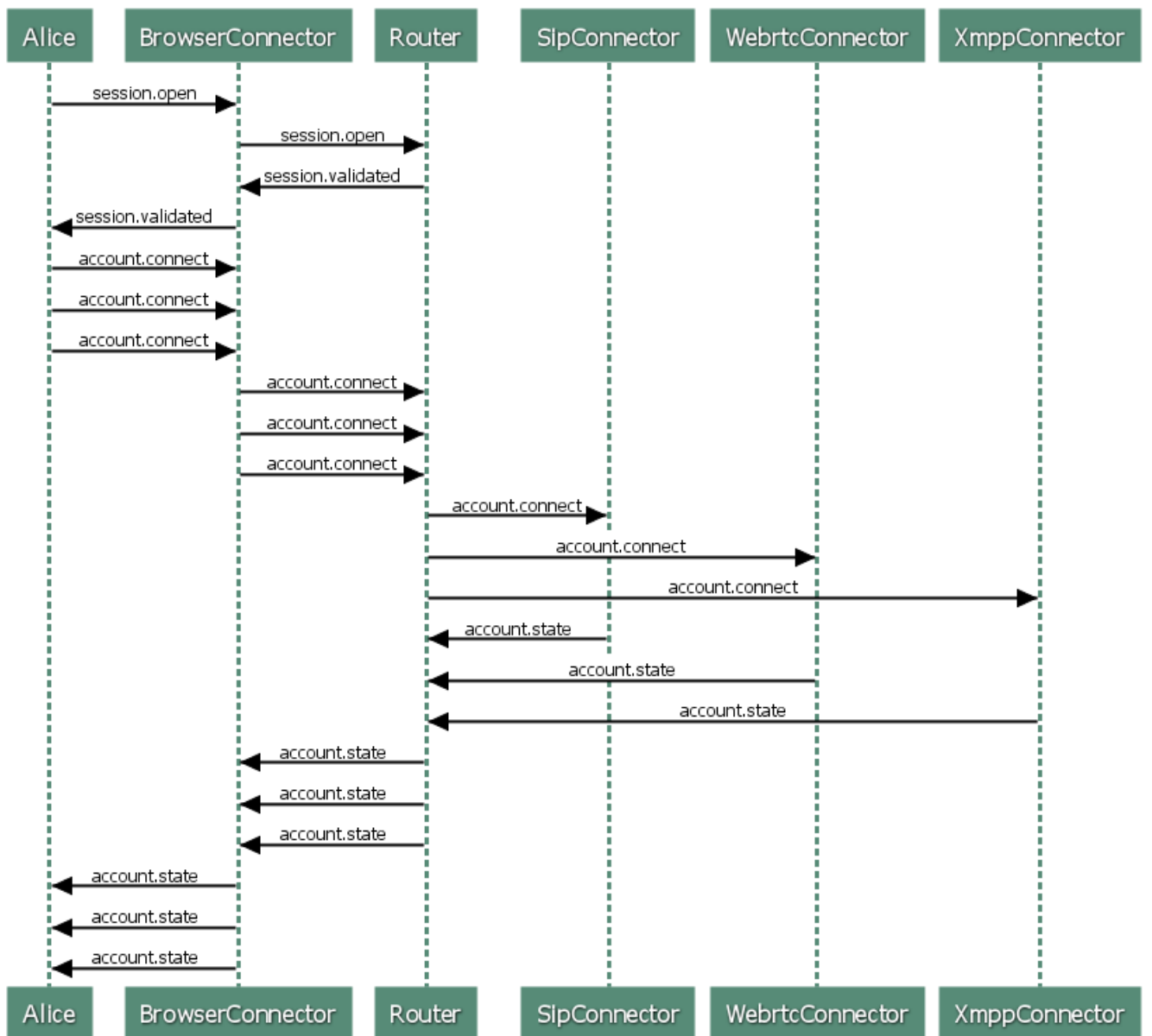
The body contains the payload of the message. Every message type has its own body schema.

### D.3.3 Account

Mainly an account consists of credentials (identifier, accessToken), that are needed to authenticate against the related network. Its lifecycle is bound to the lifecycle of the related session.

After RTC:engine received session.open, it responds a session.validated message. This message contains all provisioned accounts in its property "body.accounts".

#### D.3.3.1 Flow



www.websequencediagrams.com

### D.3.3.2 Messages

#### D.3.3.3 account.connect

RTC:engine needs one message per account. The message should contain the id of the account. The id is the object key in the accounts object from the [session.validated](../session/index.md) message.

```
{
  "from": "",
  "to": "...:...",
  "method": "account.connect",
  "session": "...",
  "body": {
    "id": "..."
  }
}
```

#### D.3.3.4 account.state

This message gives state information about the authentication and registration process of the related network and the corresponding connector. For example, if the related connector is the SipConnector, it creates a new SIP B2BUA in background, and notify the browser if any state change happens.

```
{
  "from": "...:...",
  "to": "browser:...",
  "method": "account.state",
  "session": "...",
  "body": {
    "id": "...",
    "reason": "...",
    "state": "..."
  }
}
```

#### D.3.3.5 State reasons

- OK
- CONNECTING
- DISCONNECTING
- SERVICE\_UNAVAILABLE
- SERVICE\_ERROR
- BAD\_CONFIGURATION

- WRONG\_CREDENTIALS
- CONNECTOR\_UNAVAILABLE
- CONNECTOR\_BUSY
- CONNECTOR\_ERROR
- ACCOUNT\_NOT\_FOUND

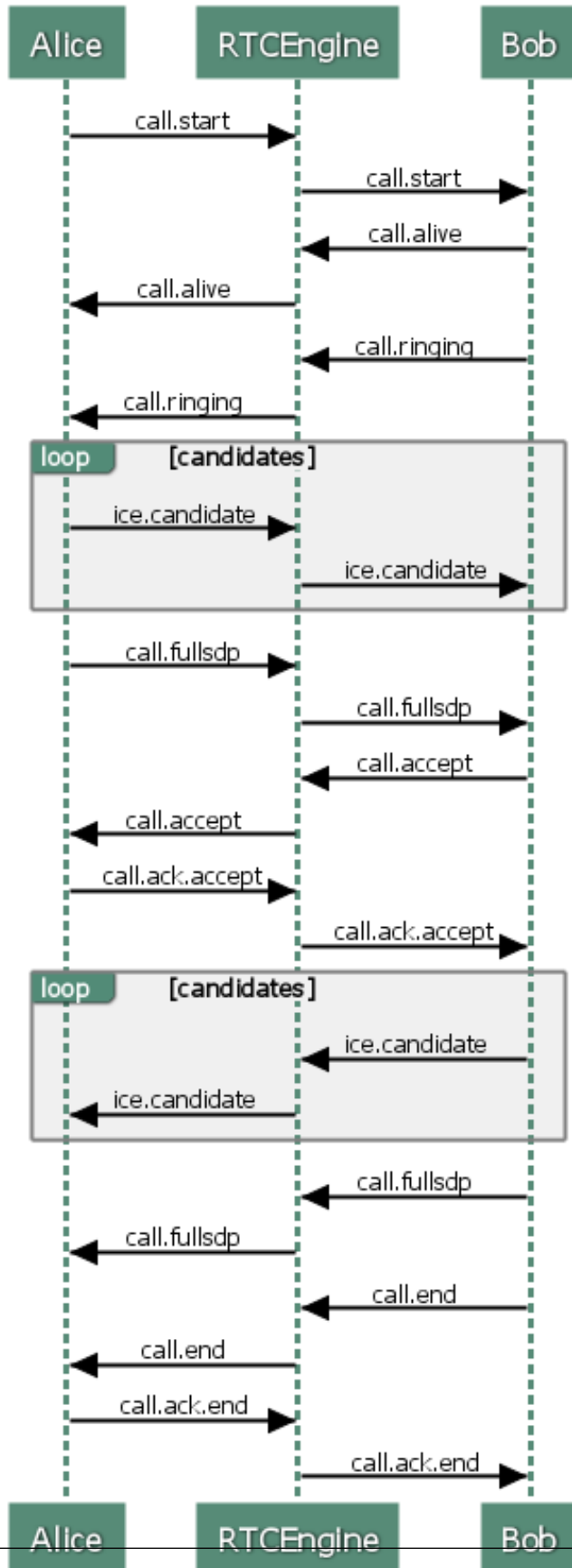
#### **D.3.3.6 States**

- CONNECTED
- DISCONNECTED



D.3.4 Call

D.3.4.1 Flow



### D.3.4.2 call.start

The caller sends this message to the RTC:engine to initiate a new call dialog.

```
{
  "from": "local",
  "to": ["...:..."],
  "method": "call.start",
  "session": "...",
  "body": {
    "id": "...",
    "gcid": "...",
    "account": "...",
    "replace": true|false,
    "trickle": true|false,
    "target": "...",
    "sdp": "..."
  }
}
```

### D.3.4.3 Body properties

#### D.3.4.4 id

The id is a UUID version 4 that identifies the call dialog in the system. But caller and callee never have the same.

#### D.3.4.5 gcid

Whereas the gcid is a system wide and end-to-end consistent call identifier. It is necessary to track the entire call dialog.

#### D.3.4.6 account

It contains the callers account id. [(See accounts)](../account/index.md)

#### D.3.4.7 replace

This property is not used yet. It should support a call handover scenario.

#### D.3.4.8 trickle

If is set to true, the callee expects ice candidates, before the full sdp delivered by the caller, to accelerate the negotiation process.

### D.3.4.9 target

It's the URI (sip:user@domain.tld) of the callee.

### D.3.4.10 sdp

The sdp property contains a very early state of the browsers media machine. It contains no ice candidates so far.

### D.3.4.11 call.alive

After the callee received the "call.start" message, it responds with a "call.alive" to the RTC:engine, immediately.

```
{
  "from": "...",
  "to": "...",
  "method": "call.alive",
  "session": "...",
  "body": {
    "id": "...",
    "gcid": "..."
  }
}
```

### D.3.4.12 call.ringing

After the callee received the "call.start" message, it responds with a "call.ringing" to the RTC:engine, immediately.

```
{
  "from": "...",
  "to": "...",
  "method": "call.ringing",
  "session": "...",
  "body": {
    "id": "...",
    "gcid": "...",
    "account": null
  }
}
```

### D.3.4.13 call.accept

The callee sends this message after accepting the call explicitly.

```
{
  "from": "...",
  "to": "...",
```



```
"method": "call.accept",
"session": "...",
"body": {
  "id": "...",
  "gcid": "...",
  "account": null,
  "trickle": true|false,
  "sdp": "..."
}
}
```

#### D.3.4.14 call.ack.accept

Caller sends this message after it received the "call.accept" message from the callee.

```
{
  "from": "...",
  "to": "...",
  "method": "call.ack.accept",
  "session": "...",
  "body": {
    "id": "...",
    "gcid": "..."
  }
}
```

#### D.3.4.15 call.candidate

Both, caller and callee send ice candidates immediately after initiating respectively accepting the call.

```
{
  "from": "...",
  "to": "...",
  "method": "call.candidate",
  "session": "...",
  "body": {
    "id": "...",
    "gcid": "...",
    "candidate": {
      "payload": "...",
      "type": "WEBRTC_LEGACY"
    }
  }
}
```

#### D.3.4.16 call.fullsdp

Both, caller and callee send this message after the ice gathering finished and all candidates are available.

```
{
  "from": "...",
  "to": "...",
  "method": "call.fullsdp",
  "session": "...",
  "body": {
    "id": "...",
    "gcid": "...",
    "sdp": "..."
  }
}
```

#### D.3.4.17 call.change....

All messages, that begin with "call.change", are important for renegotiation and glare handling.

#### D.3.4.18 call.change.lock.reset

#### D.3.4.19 call.change.lock

#### D.3.4.20 call.change.lock.ok

#### D.3.4.21 call.change.offer

#### D.3.4.22 call.change.answer

#### D.3.4.23 call.dtmf

Only works if the connector of the related account supports DTMF messages.

```
{
  "from": "...",
  "to": "...",
  "method": "call.dtmf",
  "session": "...",
  "body": {
    "id": "...",
    "gcid": "...",
    "dtmf": "...",
    "account": null
  }
}
```

#### D.3.4.24 call.end

Both, caller and callee can send this message. It forces the counter part to end and destroy the call.

```
{
  "from": "...",
  "to": "...",
  "method": "call.end",
  "session": "...",
  "body": {
    "id": "...",
    "gcid": "...",
    "reason": "..."
  }
}
```

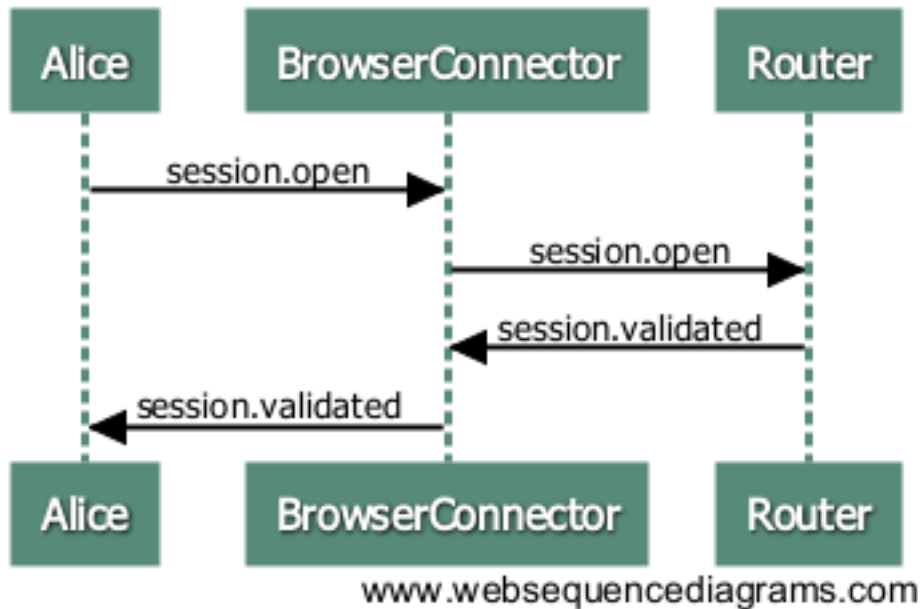
#### D.3.4.25 call.ack.end

The counter part, that receives the "call.end" message, sends the "call.ack.end" message.

```
{
  "from": "...",
  "to": "...",
  "method": "call.ack.end",
  "session": "...",
  "body": {
    "id": "...",
    "gcid": "...",
    "account": null
  }
}
```

### D.3.5 Session

#### D.3.5.1 Flow



#### D.3.5.2 Messages

##### D.3.5.3 `session.open`

```

{
  "method": "session.open",
  "from": "",
  "to": "",
  "session": "session1",
  "body": {
    "credentials": {
      "userSession": "session1"
    }
  }
}

```

##### D.3.5.4 `session.validated`

This message is the response to `session.open`. If the session property is a valid session, you get a response where the result property is true. In addition you get the account information to connect to the networks.

```

{
  "method": "session.validated",
  "from": "core",

```

```
"to": "browser:wsl",
"session": "session1"
"body": {
  "result": true,
  "accounts": {
    "account1": {
      "identifier": "sip:account1@foo.bar"
      "target": "sip-connector:b2bua-account1",
      "network": {
        "tag": "sip-network"
      }
    }
  }
},
}
```

If something went wrong, result is set to false and an error reason appears.

```
{
  "method": "session.validated",
  "from": "core",
  "to": "browser:wsl",
  "session": "session1"
  "body": {
    "result": false,
    "reason": {
      "type": "invalidToken",
      "message": "Your token is not a valid user session token!"
    }
  }
},
}
```

#### D.3.5.5 Reason types

- invalidToken
- tokenExpired
- missingCredentials

## E comx-fileshare-service

### E.1 Overview

The *comx-fileshare-service* is a Node.js (4.4.0) based filesharing service and it is intended to be used via REST API. This service allows you to upload arbitrary files to the server and to download/share them with a generated link.

The API can be used with in 2 ways:

- with **simple identification**, which means that only credentials of a user/subscriber are needed for authentication
- with **session identification**, which also provides for example the time-to-live (TTL) functionality besides authentication, and will be used in combination with the *RTC:engine*.

### E.2 Configuration and Usage

#### E.2.1 Change authentication method

To use NGCP subscribers as authentication against the API, you need to set it in the *comx-fileshare-service* config.js:

```
simpleUpload: {
  authentication: {
    enabled: true,
    subscriber: true,
    username: 'foo8',
    password: 'bar8'
  }
}
```

You can now authenticate like this with the API:

```
curl -i -X POST --insecure --form file=@/tmp/test.txt --form --user '43991002@domain.tld: ↵
x43991002'
https://$NGCP_IP/rtc/fileshare/uploads
```

If you want to use the credentials from the config.js you need so set it to the following settings:

```
simpleUpload: {
  authentication: {
    enabled: true,
    subscriber: false,
    username: 'foo8',
    password: 'bar8'
  }
}
```

In this case, the login parameter would be this:

```
curl -i -X POST --insecure --form file=@/tmp/test.txt --form --user 'foo:bar'
https://$NGCP_IP/rtc/fileshare/uploads
```

## E.2.2 Database Structure

Table information for the *fileshare* database:

- *downloads* table:

Table 28: Details of downloads Table in fileshare Database

Field Name	Field Type	Description
id	CHAR, PRIMARY KEY	Internal ID of the download action
state	ENUM	State of the download
uploaded_id	CHAR, FOREIGN KEY	External ID used for accessing the uploaded file in <i>uploads</i> table
created_at	DATETIME	Download action creation time
updated_at	DATETIME	Time of last download action modification

- *sessions* table:

Table 29: Details of sessions Table in fileshare Database

Field Name	Field Type	Description
id	CHAR, PRIMARY KEY	Internal ID of the session
ttl	INT	Time-to-live value of the session (in seconds)
created_at	DATETIME	Session creation time
updated_at	DATETIME	Time of last session modification

- *uploads* table:

Table 30: Details of uploads Table in fileshare Database

Field Name	Field Type	Description
id	CHAR, PRIMARY KEY	Internal ID of the file entry
data	LONGBLOB	The file data
original_name	VARCHAR	Original name of the file
mime_type	VARCHAR	MIME type of the file

Table 30: (continued)

Field Name	Field Type	Description
size	INT	File size in bytes
ttl	INT	Time-to-live value of the file
state	ENUM	State of the file
session_id	CHAR, FOREIGN KEY	External ID used to access session data in <i>sessions</i> table
created_at	DATETIME	File creation / upload time
updated_at	DATETIME	Time of last file modification

### E.3 Activation of Filesharing Service on NGCP

The service is installed on every sip:carrier system, but is not activated by default. In order to activate the service, connect with SSH to your server, open `/etc/ngcp-config/config.yml` with your editor of choice and change the `fileshare.enable` property from `no` to `yes`:

```
fileshare:  
  enable: yes
```

Apply the new configuration in the usual way:

```
ngcpcfg apply 'Enabled comx-fileshare-service'  
ngcpcfg push all
```

and check the status with `monit summary`. It should be now up and running.



## E.4 Message Sequence Chart

### E.4.1 Simple Message Sequence

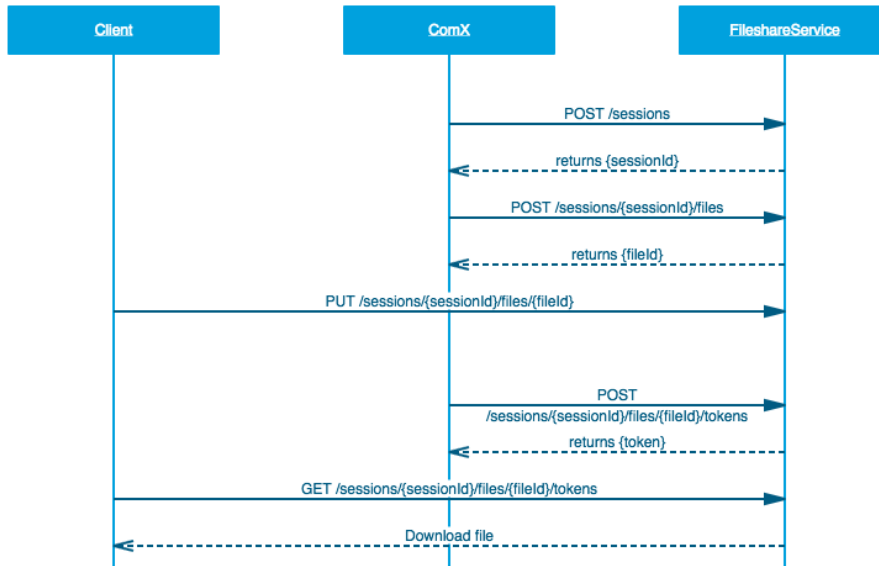


Figure 168: Sequence Simple

### E.4.2 Detailed Message Sequence

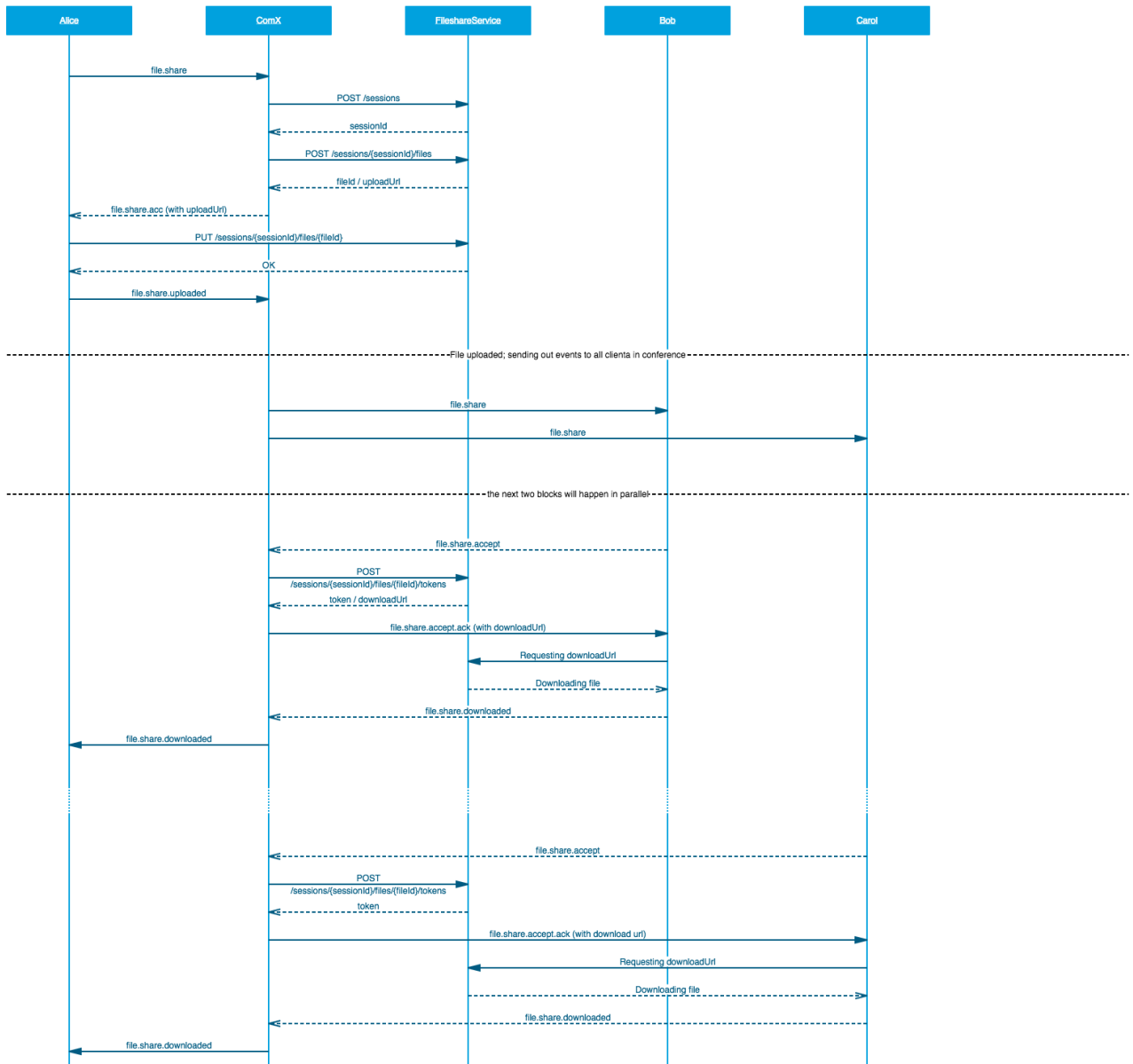


Figure 169: Sequence Detailed

## E.5 API of Filesharing Service

### E.5.1 HTTP Authentication

Type: Basic Auth  
 username/password

## E.5.2 Upload and Download with Simple Identification

The following HTTP methods can be used to perform file upload and download:

```
POST /uploads // Simple upload
GET /uploads/{fileId} // Simple download
```

## E.5.3 Upload and Download with Session Identification

The following HTTP methods can be used to perform file upload and download, and to manage sessions.

Session identification:

```
GET /sessions/{sessionId}/files // Get all files of a session
GET /sessions/{sessionId}/files/{fileId}/tokens/{tokenId} // Download a single file

POST /sessions // Create a new session
POST /sessions/{sessionId}/files // Create a new file entry
POST /sessions/{sessionId}/files/{fileId}/tokens // Generate a download token

PUT /sessions/{sessionId}/files/{fileId} // Upload and store a file
```

Simple identification:

```
GET /uploads/{fileId} // Get uploaded file
POST /uploads // Upload file
```

## E.5.4 Curl Example for Simple Upload Request

```
curl -i -X POST --insecure --form file=@/tmp/test.txt --form --user 'foo:bar'
https://$NGCP_IP/rtc/fileshare/uploads
```

## E.5.5 Upload Parameters

### E.5.5.1 file

The parameter *file* defines the path to the desired file that should be uploaded.



#### Caution

This upload parameter is mandatory!

---

Curl example:

```
curl -i -X POST --insecure --form file=@/tmp/test.txt https://$NGCP_IP/rtc/fileshare/ ↔
uploads
```

### E.5.5.2 user

The parameter *user* defines the user to authenticate with the fileshare service.



#### Caution

This upload parameter is mandatory!

---

```
curl -i -X POST --insecure --form --user 'foo:bar' https://$NGCP_IP/rtc/fileshare/uploads
```

### E.5.5.3 TTL

The parameter *ttl* defines the time-to-live (in seconds), that is how long the uploaded file will be available for download. The default values for this parameter are defined in the configuration file:

```
models: {
  session: {
    ttl: 86400 * 7
  },
  upload: {
    ttl: 3600
  }
}
```

Curl example:

```
curl -i -X POST --insecure --form file=@/tmp/test.txt --form ttl=3600
--user 'foo:bar' https://$NGCP_IP/rtc/fileshare/uploads
```

Response from *curl* when TTL is expired:

```
{
  "message": "upload expired"
}
```

Response in the log file when TTL is expired:

```
Error at /uploads/88e5905d-5d96-4750-ab3d-77a1ed26f569: message=upload expired, status=410
```

### E.5.6 Number of Possible Downloads

There is a significant difference in the usage of the filesharing service between the approach within the *RTC:engine* and the simple upload/download one:

- If you are using the **simple upload and download** approach, the generated download link you get for your file can be used as many times as required, as long as the TTL is not expired.
- The approach **with the Session ID**, which will be used with the *RTC:engine* implementation, limits the download to one-time only. This means that the generated download link can be used only once. If you plan to share the URL with multiple persons, you have to generate one link for each recipient.

## F NGCP Internals

This chapter documents internals of the sip:carrier that should not be usually needed, but might be helpful to understand the overall system.

### F.1 Pending reboot marker

The sip:carrier has the ability to mark a pending reboot for any server, using the file `/var/run/reboot-required`. As soon as the file exists, several components will report about a pending reboot to the end-user. The following components report about a pending reboot right now: `ngcp-status`, `ngcpcfg status`, `motd`, `ngcp-upgrade`. Also, `ngcp-upgrade` will NOT allow proceeding with an upgrade if it notices a pending reboot. It might affect `rtengine` dkms module building if there is a pending reboot requested by a newly installed kernel, etc.

### F.2 Redis id constants

The list of current sip:carrier Redis DB IDs:

Service	Redis DB N:	central (role db)	local	Release	Ticket	Description
<b>sems</b>	redis_db:	-	0	mr3.7.1+	-	HA switchover
<b>rtengine</b>	redis_db:	-	1	mr3.7.1+	-	HA switchover
<b>proxy</b>	redis_db:	2	-	mr3.7.1+	-	Counter of hunting groups
<b>proxy</b>	redis_db:	3	-	mr3.7.1+	-	Concurrent dialog counters
<b>proxy</b>	redis_db:	-	4	mr3.7.1+	-	List of keys of the central counters
<b>prosody</b>	redis_db:	5	-	mr3.7.1+	-	XMPP cluster
<b>sems PBX</b>	redis_db:	-	6	mr3.7.1+	-	HA switchover
<b>sems</b>	redis_db:	7	-	mr4.1.1+	MT#12707	Sems malicious_call app
<b>captagent</b>	redis_db:	-	8	mr4.1.1+	MT#15427	Captagent internal data
<b>monitoring</b>	redis_db:	9	-	mr4.3+	MT#31	Old SNMP agent monitoring data (unused)
<b>proxy</b>	redis_db:	10	-	mr4.3+	MT#16079	SIP Loop detection

### F.2.1 InfluxDB monitoring keys

The *InfluxDB ngcp* monitoring database contains time series of several monitoring sources. The following are some of the current measurements:

node	Cluster node information.
memory	System memory information.
proc_count	Process counts.
monit	Monit supervised processes information.
mail	MTA information.
mysql	MySQL database information.
kamailio	Kamailio statistics information.
sip	SIP statistics information.

The *node* measurement contains the following fields:

active	Cluster node HA state (boolean: 1/0).
hb_proc_state	Cluster node heartbeat process state (boolean: stopped/running).
hb_host_state	Cluster node host state (boolean: up/down).
hb_node_state	Cluster node HA state (ngcp-check-active -p).

The *monit* measurement contains the following fields:

name	The process name.
proc_status	The process status.
monit_status	The monit status.
pid	The process ID.
ppid	The process parent ID.
children	The number of children.
uptime	The process uptime.
cpu_percent	The CPU usage in percent for this process.
cpu_percent_total	The CPU usage in percent for the process group.
memory	The memory in bytes for this process.
memory_total	The memory in bytes for the process group.
memory_percent	The memory in percent for this process.
memory_percent_total	The memory in percent for the process group.
data_collected	The timestamp when the data was collected.

The *mysql* measurement contains the following fields:

last_io_error	Last IO error description.
last_sql_error	Last SQL error description.
queries_per_second_average	Average of queries per second.

replication_discrepancies	Number of replication discrepancies.
---------------------------	--------------------------------------

### F.3 Enum preferences

All tables are in database "provisioning".

So called "enum preferences" allow a fixed set of possible values, an enumeration, for preferences. Following the differences between other preferences are described.

Setting the attribute "data\_type" of table "voip\_preferences" to "enum" marks a preferences as an enum. The list of possible options is stored in table "voip\_preferences\_enum".

voip\_preferences\_enum is:

id

boring pkey

preference\_id

Reference to table voip\_preferences.

label

A label to be displayed in frontends.

value

Value that will be written to voip\_[usr|dom|peer]\_preferences.value if it is NOT NULL. Will not be written if it IS NULL. This can be used to implement a "default value" for a preference that is visible in frontends as such (will be listed first if nothing is actually selected), but will not be written to voip\_[usr|dom|peer]\_preferences.value. Usually forcing a domain or peer default. Should also be named clearly (eg. \_\_"use domain default"\_\_). (Note: Therefore will also not be written to any kamailio table.)

usr\_pref

dom\_pref

peer\_pref

Flag if this is to be used for [usr|dom|peer] preferences.

default\_val

Flag indicating if this should be used as a default value when creating new entities or introducing new enum preferences (both done via triggers). (Note: For this to work, value must also be set.)

Relevant triggers:



`enum_update`

Propagates changes of `voip_preferences_enum.value` to `voip_[usr|dom|peer]_preferences.value`

`enum_set_default`

Will create entries for default values when adding a new enum preference. The default value is the tuple from `voip_preferences_enum` WHERE `default_val=1` AND `value NOT NULL`.

`trigger voip_dom_crepl_trig``trigger voip_phost_crepl_trig``trigger voip_sub_crepl_trig`

These three triggers will set possible default values (same condition as for `enum_set_default`) when creating new subscribers/domains/peers.

Find a usage example in a section in *db-schema/db\_scripts/diff/9086.up*.