The sip:provider CE Handbook 2.4.1

Contents

1 Introduction

1.1 About this Document

This document describes the architecture and the operational steps to install, operate and modify the Sipwise sip:provider CE.

The first chapter gives an introduction into the sip:provider CE. It describes what it is, what it contains and who should use it.

The second chapter guides through the installation and upgrade process of the sip:provider CE. It lines out the prerequisites and the steps required to install it from scratch or upgrade it from an older version.

The third chapter provides an architectural overview of what software components are used within the sip:provider CE. It goes into each of the main components and outlines how the signaling and media paths are routed through the system. It also provides charts of basic call flows to give you an idea how the system internally works.

The fourth chapter describes the steps to configure the sip:provider CE in order to offer VoIP services to end users.

The fifth chapter shows the different customer self-care interfaces and describes how to configure them.

The sixth chapter describes the billing interface, so you can rate calls and export call detail records.

The seventh chapter describes in detail the steps necessary if you want to start modifying local configuration files to adapt and extend the system.

The eight chapter guides through the provisioning interface (SOAP and XMLRCP) system, so you can integrate it into your own or third-party provisioning and billing systems.

The ninth chapter outlines additional tasks to protect the sip:provider CE from security exploits and attacks, as well as describes the steps to perform regular backups and restore from them when necessary.

1.2 Getting Help

1.2.1 Community Support

We have set up the *spce-user* mailing list, where questions are answered on a best-effort basis and discussions can be started with other community users.

1.2.2 Commercial Support

If you need professional help setting up and maintaining the sip:provider CE, send an email to support@sipwise.com.

Sipwise also provides training and commercial support for the platform. Additionally, we offer a migration path to the sip:provider PRO appliance, which is the commercial, carrier-grade version of the sip:provider CE. If the user base grows on the CE, this will allow operators to migrate seamlessly to a highly available and scalable platform with defined service level agreements, phone support and on-call duty. Please visit www.sipwise.com for more information on commercial offerings.

1.3 What is the sip:provider CE?

The sip:provider CE 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 SOAP and XMLRPC APIs.

The sip:provider CE can be installed in a few steps within a couple of minutes and requires no knowledge about configuration files of specific software components.

1.4 What is inside the sip:provider CE?

Opposed to other free VoIP software, the sip:provider CE 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, Apache, Catalyst, Kamailio, SEMS, Asterisk etc. as its core building blocks. These blocks are glued together using optimized and proven configurations and work-flows and are complemented by building blocks developed by Sipwise to provide fully-featured and easy to operate VoIP services.

After downloading and starting the installer, it will fetch and install all the required Debian packages from the relevant Debian repositories. The installed applications are managed by the NGCP Configuration Framework, which allows 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:provider CE. By using the provided provisioning and billing APIs, it can be integrated tightly into existing OSS/BSS infrastructures to optimize work-flows.

1.5 Who should use the sip:provider CE?

The sip:provider CE is specifically tailored to companies and engineers trying to start or experiment with a 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 and implementing the missing components to build a business on top of that.

In the past, creating a business-ready VoIP service included installation and configuration of SIP software like Asterisk, OpenSER, Kamailio etc., which can get quite difficult when it comes to implementing advanced features. It required to implement different web interfaces, billing engines and connectors to existing OSS/BSS infrastructure. These things are now obsolete due to the CE, which covers all these requirements.

2 Installation and Upgrade

2.1 Initial Installation

2.1.1 Prerequisites

For an initial installation of the sip:provider CE, it is mandatory that your production environment meets the following criteria:

HARDWARE REQUIREMENTS

- Recommended: Dual-core, x86_64 compatible, 3GHz, 4GB RAM, 128GB HDD
- Minimum: Single-core, x86_64 compatible, 1GHz, 1GB RAM, 16GB HDD

SUPPORTED OPERATING SYSTEMS

- Debian Squeeze (6.0) 64-bit

INTERNET CONNECTION

- Hardware needs connection to the Internet

Important
Only **Debian Squeeze (6.0) 64-bit** is currently supported as a host system for the sip:provider CE.

Important
It is **HIGHLY** recommended that you use a **dedicated server** (either a physical or a virtual one) for sip:provider CE, because the installation process will wipe out existing MySQL databases and modify several system configurations.

2.1.2 Using the NGCP installer (recommended)

Installing the Operating System

You need to install Debian Squeeze (6.0) 64-bit on the server. A **basic** installation without any additional task selection (like *Desktop System*, *Web Server* etc.) is sufficient.

Tip
Sipwise recommends using the Netinstall ISO (md5sum) as installation medium.

Important
If you use other kinds of installation media (e.g. provided by your hosting provider), prepare for some issues that might come up during installation. For example, you might be forced to manually resolve package dependencies in order to install the sip:provider CE. Therefore, it is HIGHLY RECOMMENDED to use a clean Debian installation to simplify the installation process.

If you plan to install the sip:provider CE on Virtual Hosting Providers like *Dreamhost* with their provided Debian installer, you might need to manually prepare the system for the NGCP installation, otherwise the installer will fail installing certain package versions required to function properly.

Using Dreamhost Virtual Private Server

A Dreamhost virtual server uses apt-pinning and installs specific versions of MySQL and apache, so you need to clean this up beforehand.

```
apt-get remove --purge mysql-common ndn-apache22
mv /etc/apt/preferences /etc/apt/preferences.bak
apt-get update
apt-get dist-upgrade
```

Warning
Be aware that this step will break your web-based system administration provided by Dreamhost. Only do it if you are certain that you won't need it.

Installing the sip:provider CE

The sip:provider CE is based on the *Sipwise NGCP*, so download and install the *Sipwise NGCP* installer package:

```
PKG=ngcp-installer-latest.deb
wget http://deb.sipwise.com/spce/${PKG}
dpkg -i ${PKG}
```

Run the installer as root user:

```
ngcp-installer
```

The installer will ask you to confirm that you want to start the installation. Read the given information **carefully**, and if you agree, proceed with *y*.

The installation process will take several minutes, depending on your network connection and server performance. If everything goes well, the installer will (depending on the language you use), show something like this:

```
Installation finished. Thanks for choosing NGCP sip:provider Community Edition.
```

During the installation, you can watch the background processing by executing the following command on a separate console:

```
tail -f /tmp/ngcp-installer.log
```

2.1.3 Using a pre-installed virtual machine

For quick test deployments, pre-installed virtualization images are provided. These images are intended to be used for quick test, not recommended for production use.

VirtualBox image

You can download a VirtualBox image from here (checksums: sha1, md5). Once you have downloaded the file you can import it to VirtualBox via its import utility.

The format of the image is *ova*. If you have VirtualBox 3.x running, which is not compatible with *ova* format, you need to extract the file with any *tar* compatible software and import the *ovf* file which is inside the archive.

On Linux, you can do it like this:

```
tar xvf sip_provider_CE_2.4.1_virtualbox.ova
```

On Windows, right-click on the ova file, choose *Open with* and select *WinZIP* or *WinRAR* or any other application able to extract *tar* archives. Extract the files to any place and import the resulting *ovf* file in VirtualBox.

Considerations when using this virtual machine:

- You will need a 64bit guest capable VirtualBox setup.
- The root password is *sipwise*
- There's a user *sipwise* with password *sipwise*
- You should use *bridge mode* networking (adjust your bridging interface in the virtual machine configuration) to avoid having the sip:provider CE behind NAT.
- You'll need to adjust your timezone and keyboard layout.
- The network configuration is set to DHCP. You'll need to change it to the appropriate static configuration.
- As the virtual image is a static file, it won't contain the most updated versions of our software. Please upgrade the system via apt as soon as you boot it for the first time.

VMware image

You can download a VMware image from here (checksums: sha1, md5). Once you have downloaded the file just extract the *zip* file and copy its content to your virtual machines folder.

Considerations when using this virtual machine:

- You will need a 64bit guest capable vmware setup.
- The root password is *sipwise*

- There’s a user *sipwise* with password *sipwise*
- You’ll need to adjust your timezone and keyboard layout.
- The network configuration is set to DHCP. You’ll need to change it to the appropriate static configuration.
- As the virtual image is a static file, it won’t contain the most updated versions of our software. Please upgrade the system via apt as soon as you boot it for the first time.

2.2 Initial System Configuration

After the installation went through successfully, you are ready to adapt the system parameters to your needs to make the system work properly.

2.2.1 Network Configuration

The only parameter you need to change at this moment is the listening address for your SIP services. To do this, modify the parameter *networking→eaddress* in */etc/ngcp-config/config.yml*, which by default is set to *127.0.0.1*:

```
vim /etc/ngcp-config/config.yml
```

Look for the following section on top of the file:

```
networking:
  eaddress: 127.0.0.1

[...]
```

Change this parameter to the IP address configured during install time of the Debian operating system. If you haven’t fully configured your network interfaces, do this by adapting also the file */etc/network/interfaces*:

```
vim /etc/network/interfaces
```

Add or adapt your interface configuration accordingly. For example, if you just want to use the system in your internal network 192.168.0.0/24, it could look something like this:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
        address 1.2.3.4
        netmask 255.255.255.0
        gateway 1.2.3.1
        dns-nameservers 8.8.8.8
        dns-search yourdomain.com
```

```
/etc/init.d/networking restart
```

Warning
It is **HIGHLY** recommended that you use a public IP for the system. If you use a private IP, all your subscribers and gateways must be in the same private network. The sip:provider CE handles gateways and subscribers behind NAT, but the sip:provider CE itself cannot be behind NAT.

2.2.2 Apply Configuration Changes

In order to apply the changes you made to */etc/ngcp-config/config.yml*, you need to execute the following command to re-generate your configuration files and to automatically restart the services:

```
ngcpcfg apply
```

Tip
At this point, your system is ready to serve.

2.2.3 Start Securing Your Server

During installation, the system user *cdrexport* is created. This jailed system account is supposed to be used to export CDR files via sftp/scp. Set a password for this user by executing the following command:

```
passwd cdrexport
```

The installer has set up a MySQL database on your server. You need to set a password for the MySQL root user to protect it from unauthorized access by executing this command:

```
mysqladmin password <your mysql root password>
```

For the *Administrative Web Panel* located at *https://<your-server-ip>:1443/*, a default user *administrator* with password *administrator* has been created. Connect to the panel (accept the SSL certificate for now) using this credentials and change the password of this user by going to *System Administration→Administrators* and clicking *edit*.

2.2.4 Configuring the Email Server

The NGCP installer will install *mailx* (which has *Exim4* as MTA as a default dependency) on the system, however the MTA is not configured by the installer. If you want to use the *Voicemail-to-Email* feature of the Voicebox, you need to configure your MTA properly. If you are fine to use the default MTA *Exim4*, execute the following command:

```
dpkg-reconfigure exim4-config
```

Depending on your mail setup in your environment (whether to use a smarthost or not), configure Exim accordingly. In the most simple setup, apply the following options when prompted for it:

- **General type of mail configuration:** `internet site; mail is sent and received directly using SMTP`
- **System mail name:** the FQDN of your server, e.g. `ce.yourdomain.com`
- **IP-addresses to listen on for incoming SMTP connections:** `127.0.0.1`
- **Other destinations for which mail is accepted:** the FQDN of your server, e.g. `ce.yourdomain.com`
- **Domains to relay mail for:** leave empty
- **Machines to relay mail for:** leave empty
- **Keep number of DNS-queries minimal (Dial-on-Demand)?** `No`
- **Delivery method for local mail:** `mbox format in /var/mail/`
- **Split configuration into small files?** `No`

Important
You are free to install and configure any other MTA (e.g. postfix) on the system, if you are more comfortable with that.

2.2.5 What’s next?

To test and use your installation, you need to follow these steps now:

1. Create a SIP domain
2. Create some SIP subscribers
3. Register SIP endpoints to the system
4. Make local calls and test subscriber features
5. Establish a SIP peering to make PSTN calls

Please read the next chapter for instructions on how to do this.

2.3 Upgrade from v2.2 to v2.4

The system upgrade from sip:provider CE v2.2 to v2.4 will perform a couple of fundamental tasks:

1. Convert the previous implementation with several rewrite rules on the peer/domain to the rewrite rule sets assigned to peer/user/domain as preferences.
2. Convert the kamailio and accounting databases to UTF-8.
3. Convert the CDR and accounting timestamp field to data type with milliseconds without loss of the old data.

For upgrading the sip:provider CE from v2.2 to the latest v2.4 release, execute the following commands:

```
apt-get update
apt-get install ngcp-upgrade-2.4-ce
```

Run the upgrade script as *root* like this:

```
ngcp-upgrade
```

The upgrade script will ask you to confirm that you want to start. Read the given information **carefully**, and if you agree, proceed with *y*.

The upgrade process will take several minutes, depending on your network connection and server performance. If everything goes well, it will finally ask you to reboot your system. Confirm to let the system reboot (it will boot with an updated kernel).

Once up again, double-check your config file */etc/ngcp-config/config.yml* (sections will be rearranged now and will contain more parameters) and your domain/subscriber/peer configuration and test the setup. You can find a backup of some important configuration files of your existing 2.2 installation under */var/backup/ngcp-2.4/* in case you need to roll back something at any time.

3 Platform Architecture

The sip:provider CE platform is one single node running all necessary components of the system. The components are outlined in the following figure:

Architecture Overview

The main building blocks of the sip:provider CE are:

- SIP Signaling and Media Relay
- Provisioning
- Mediation and Billing

3.1 SIP Signaling and Media Relay

In SIP-based communication networks, it is important to understand that the signaling path (e.g. for call setup and tear-down) is completely independent of the media path. On the signaling path, the involved endpoints negotiate the call routing (which user calls which endpoint, and via which path - e.g. using SIP peerings or going through the PSTN - the call is established) as well as the media attributes (via which IPs/ports are media streams sent and which capabilities do these streams have - e.g. video using H.261 or Fax using T.38 or plain voice using G.711). Once the negotiation on signaling level is done, the endpoints start to send their media streams via the negotiated paths.

3.1.1 SIP and Media Elements

The components involved in SIP and Media on the sip:provider CE are shown in the following figure:

SIP and Media Relay Components

SIP Load-Balancer

The SIP load-balancer is a Kamailio instance acting as ingress and egress point for all SIP traffic to and from the system. It's a high-performance SIP proxy instance based on Kamailio and is responsible for sanity checks of inbound SIP traffic. It filters broken SIP messages, rejects loops and relay attempts and detects denial-of-service and brute-force attacks and gracefully handles them to protect the underlying SIP elements. It also performs the conversion of TLS to internal UDP and vice versa for secure signaling between endpoints and the sip:provider CE, and does far-end NAT traversal in order to enable signaling through NAT devices.

The load-balancer is the only SIP element in the system which exposes a SIP interface to the public network. Its second leg binds in the switch-internal network to pass traffic from the public internet to the corresponding internal components.

The name load-balancer comes from the fact that in the commercial version, when scaling out the system beyond just one pair of servers, the load-balancer instance becomes its own physical node and then handles multiple pairs of proxies behind it.

On the public interface, the load-balancer listens on port 5060 for UDP and TCP, as well as on 5061 for TLS connections. On the internal interface, it speaks SIP via UDP on port 5060 to the other system components, and listens for XMLRPC connections on TCP port 5060, which is used by the OSSBSS system to control the daemon.

Its config files reside in `/etc/ngcp-config/templates/etc/kamailio/lb/`, and changes to these files are applied by executing `ngcpcfg apply`.

Tip
The SIP load-balancer can be managed via the commands `/etc/init.d/kamailio-lb start`, `/etc/init.d/kamailio-lb stop` and `/etc/init.d/kamailio-lb restart`. Its status can be queried by executing `/etc/init.d/kamailio-lb status`.

SIP Proxy/Registrar

The SIP proxy/registrar (or short *proxy*) is the work-horse of the sip:provider CE. It's also a separate Kamailio instance running in the switch-internal network and is connected to the provisioning database via MySQL, authenticates the endpoints, handles their registrations on the system and does the call routing based on the provisioning data. For each call, the proxy looks up the provisioned features of both the calling and the called party (either subscriber or domain features if it's a local caller and/or callee, or peering features if it's from/to an external endpoint) and acts accordingly, e.g. by checking if the call is blocked, by placing call-forwards if applicable and by normalizing numbers into the appropriate format, depending on the source and destination of a call.

It also writes start- and stop-records for each call, which are then transformed into call detail records (CDR) by the mediation system.

If the endpoints indicate negotiation of one or more media streams, the proxy also interacts with the *Media Relay* to open, change and close port pairs for relaying media streams over the sip:provider CE, which is especially important to traverse NAT.

The proxy listens on UDP port 5062 in the system-internal network. It cannot be reached directly from the outside, but only via the SIP load-balancer.

Its config files reside in `/etc/ngcp-config/templates/etc/kamailio/proxy/`, and changes to these files are applied by executing `ngcpcfg apply`.

Tip
The SIP proxy can be controlled via the commands `/etc/init.d/kamailio-proxy start`, `/etc/init.d/kamailio-proxy stop` and `/etc/init.d/kamailio-proxy restart`. Its status can be queried by executing `/etc/init.d/kamailio-proxy status`.

SIP Back-to-Back User-Agent (B2BUA)

The SIP B2BUA (also called SBC within the system) decouples the first call-leg (calling party to sip:provider CE) from the second call-leg (sip:provider CE to the called party).

The software part used for this element is SEMS.

This element is typically optional in SIP systems, but it is always used for SIP calls (INVITE) that don't have the sip:provider CE as endpoint. It acts as application server for various scenarios (e.g. for feature provisioning via Vertical Service Codes and as Conferencing Server) and performs the B2BUA decoupling, topology hiding, caller information hiding, SIP header and Media feature filtering, outbound registration, outbound authentication and call length limitation as well as Session Keep-Alive handler.

Due to the fact that typical SIP proxies (like the load-balancer and proxy in the sip:provider CE) do only interfere with the content of SIP messages where it's necessary for the SIP routing, but otherwise leave the message intact as received from the endpoints, whereas the B2BUA creates a new call leg with a new SIP message from scratch towards the called party, SIP message sizes are reduced significantly by the B2BUA. This helps to bring the message size under 1500 bytes (which is a typical default value for the MTU size) when it leaves the sip:provider CE. That way, chances of packet fragmentation are quite low, which reduces the risk of running into issues with low-cost SOHO routers at customer sides, which typically have problems with UDP packet fragmentation.

The SIP B2BUA only binds to the system-internal network and listens on UDP port 5080 for SIP messages from the load-balancer or the proxy, on UDP port 5040 for control messages from the cli tool and on TCP port 8090 for XMLRPC connections from the OSSBSS to control the daemon.

Its configuration files reside in `/etc/ngcp-config/templates/etc/sems`, and changes to these files are applied by executing `ngcpcfg apply`.

Tip
The SIP B2BUA can be controlled via the commands `/etc/init.d/ngcp-sems start`, `/etc/init.d/ngcp-sems stop` and `/etc/init.d/ngcp-sems restart`. Its status can be queried by executing `/etc/init.d/ngcp-sems status`

SIP App-Server

The SIP App-Server is an Asterisk instance used for voice applications like Voicemail and Reminder Calls. Asterisk uses the MySQL database as a message spool for voicemail, so it doesn't directly access the file system for user data. The voicemail plugin is a slightly patched version based on Asterisk 1.4 to make Asterisk aware of the sip:provider CE internal UUIDs for each subscriber. That way a SIP subscriber can have multiple E164 phone numbers, but all of them terminate in the same voicebox.

The App-Server listens on the internal interface on UDP port 5070 for SIP messages and by default uses media ports in the range from UDP port 10000 to 20000.

The configuration files reside in `/etc/ngcp-config/templates/etc/asterisk`, and changes to these files are applied by executing `ngcpcfg apply`.

Tip
The SIP App-Server can be controlled via the commands `/etc/init.d/asterisk start`, `/etc/init.d/asterisk stop` and `/etc/init.d/asterisk restart`. Its status can be queried by executing `/etc/init.d/asterisk status`

Media Relay

The Media Relay (also called *mediaproxy-ng* or *mediaproxy*) is a Kernel-based packet relay, which is controlled by the SIP proxy. For each media stream (e.g. a voice and/or video stream), it maintains a pair of ports in the range of port number 30000 to 40000. When the media streams are negotiated, mediaproxy opens the ports in user-space and starts relaying the packets to the addresses announced by the endpoints. If packets arrive from different source addresses than announced in the SDP body of the SIP message (e.g. in case of NAT), the source address is implicitly changed to the address the packets are received from. Once the call is established and the mediaproxy has received media packets from both endpoints for this call, the media stream is pushed into the kernel and is then handled by a custom Sipwise iptables module to increase the throughput of the system and to reduce the latency of media packets.

The mediaproxy internally listens on UDP port 12222 for control messages from the SIP proxy. For each media stream, it opens two pairs of UDP ports on the public interface in the range of 30000 and 40000 per default, one pair on odd port numbers for the media data, and one pair on the next even port numbers for meta data, e.g. RTCP in case of RTP streams. Each endpoint communicates with one dedicated port per media stream (opposed to some implementations which use one pair for both endpoints) to avoid issues in determining where to send a packet to. The mediaproxy also sets the QoS/ToS/DSCP field of each IP packet it sends to a configured value, 184 (0xB8, *expedited forwarding*) by default.

The kernel-internal part of the mediaproxy is facilitated through an *iptables* module having the target name `MEDIAPROXY`. If any additional firewall or packet filtering rules are installed, it is imperative that this rule remains untouched and stays in place. Otherwise, if the rule is removed from iptables, the kernel will not be able to forward the media packets and forwarding will fall back to the user-space daemon. The packets will still be forwarded normally, but performance will be much worse under those circumstances, which will be especially noticeable when a lot of media streams are active concurrently. See the section on *Firewalling* for more information.

The mediaproxy configuration file is `/etc/ngcp-config/templates/etc/default/ngcp-mediaproxy-ng-daemon`, and changes to this file are applied by executing `ngcpcfg apply`. The UDP port range can be configured via the `config.yml` file under the section `rtpproxy`. The QoS/ToS value can be changed via the key `qos.tos_rtp`.

Tip
The Media Relay can be controlled via the commands `/etc/init.d/ngcp-mediaproxy-ng-daemon start`, `/etc/init.d/ngcp-mediaproxy-ng-daemon stop` and `/etc/init.d/ngcp-mediaproxy-ng-daemon restart`. Its status can be queried by executing `/etc/init.d/ngcp-mediaproxy-ng-daemon status`

3.1.2 Basic Call Flows

Endpoint Registration

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 the reach a subscriber in case on 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).

By default, a subscriber can register 5 contacts for an Address of Record (AoR, e.g. `sip:someuser@example.org`).

Basic Call

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 outer (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.

Session Keep-Alive

Session-Timer Call-Flow

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 after a default of 150 seconds to check if the endpoints are still alive and responsive. Both endpoints can renegotiate the timer within a range of 45 seconds and 3600 seconds by default. All values can be tuned in the sip:provider CE configuration.

Tip

Keep in mind that the values being used in the signaling are always half the value being configured. So in the configuration, you will find 300 seconds, 90 seconds and 7200 seconds for the values given above, because the Session-Timer standard suggests a refresh rate of "sessiontimer/2".

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.

Voicebox Calls

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:provider CE 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.

4 Administrative Configuration

To be able to configure your first test clients, you will need a SIP domain and some subscribers in this domain. Throughout this steps, let's assume you're running the NGCP on the IP address *1.2.3.4*, and you want this IP to be used as SIP domain. This means that your subscribers will have an URI like *user1@1.2.3.4*.

Tip
You can of course set up a DNS name for your IP address (e.g. letting sip.yourdomain.com point to 1.2.3.4) and use this DNS name throughout the next steps, but we'll keep it simple and stick directly with the IP as a SIP domain for now.

Warning
Once you started adding subscribers to a SIP domain, and later decide to change the domain, e.g. from 1.2.3.4 to sip.yourdomain.com, you'll need to recreate all your subscribers in this new domain. It's currently not possible to easily change the domain part of a subscriber.

Go to the *Administrative Web Panel* (*Admin Panel*) running on *https://<ce-ip>:1443/* and follow the steps below. The default user on the system is *administrator* with the password *administrator*, if you haven't changed it already in [?simpara].

4.1 Creating Domains

Go to *System Administration→Domains*. You'll see a form *Create Domain*, where you have to provide the name of your SIP domain. In our example we'll put in *1.2.3.4* there and click *add*. The newly created domain now shows up under *Edit Domains*.

Tip
You will most likely want to assign Rewrite Rule Set to your domain via Preferences tab. The usage of Rewrite Rule Sets is explained in Section 4.5.

4.2 Creating Accounts

An *account* on the NGCP is a billing container, which contains one or more subscribers for a customer. In this billing container, you can define which *Billing Profile* is used for calls being placed by the subscribers of this account.

To create a new account, go to *User Administration→Accounts* and click *Create new account*. For our first tests, we will use the default values, so just click *Save*.

You will be presented with an overview of the new account, showing basic *Account Information*, the *Account Balance* (which will only get relevant when you start using your own Billing Profile) and the list of *Subscribers* for this account, which is currently empty.

4.3 Creating Subscribers

In the *Subscribers* section at the bottom of the account information for the account you created before, click *create new* to create a new subscriber for this account. You will be presented with the *Master Data* form, where you have to fill in the following options:

- **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.
- **E.164 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 a made-up 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 phantasy 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 4.5.

Important
NGCP allows single subscriber to have multiple E.164 numbers to be used as aliases for receiving incoming calls. Also NGCP supports "implied" extensions, e.g. if a subscriber has number 012345, but somebody calls 012345100, then it first tries to send the call to number 012345100 (even though the user is registered as myusername), and only after 404 it falls back to the user-part for which the user is registered.

Tip
The interface ensures that any numbers entered conform to the local numbering plan. The numbering plan is defined by a set of three regular expressions, which can be found and edited in `config.yml` under the key `ossbss.provisioning.routing`. The defaults should work for most setups, but under some circumstances (e.g. there are no area codes) it may be necessary to customize these.

- **SIP URI**: Insert the user part of the URI into the first field (e.g. *user1*) and select the domain you want to put this subscriber into in the drop-down.

Caution
With the default system settings, the user part has to have at least one alphabetic character, so it's not possible by default to just use a number here. To allow that, on the console set *ossbss→provisioning→allow_numeric_usernames* to *1* in */etc/ngcp-config/config.yml* and execute the command `ngcpcfg apply`. If you want for example to set a numeric customer id as the SIP user, make sure it does not overlap with actual phone numbers, otherwise these calls will be routed to the SIP user instead of the phone number.

- **SIP password**: The password of your subscriber to authenticate on the SIP proxy. It must be at least 6 characters long.
- **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*.

Click *Save* to create the subscriber. Repeat the creation of accounts and subscribers for all your test accounts. You should have at least 3 subscribers to test all 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.

At some point, you should revise the subscriber Preferences, in particular the CLI options that 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 also.
- You can override any user-provided number coming from the subscriber using the *user_cli* preference.

4.4 Creating Peerings

If you want to terminate calls at or allow calls from 3rd party systems (e.g. PSTN gateways, SIP trunks), you need to create SIP peerings for that. To do so, go to *System Administration→SIP Peerings*. There you can add peering groups, and for each peering group add peering servers. Every peering group needs a peering contract for correct interconnection billing.

4.4.1 Creating Peering Contracts

In order to create peering groups, you must create at least one peering contract. It defines, which billing profile is going to be used for calls to the corresponding peering groups. In this example, we will use the *Default Billing Profile*, because we haven't set up proper profiles yet.

In the *SIP Peering Contracts* section, click *create new* to add a peering contract. You will be presented with a form to set the billing profile and some contact details for the contract. To create a very basic dummy profile, we will set the following values:

- **billing profile:** `Default Billing Profile`
- **First Name:** leave empty
- **Last Name:** leave empty
- **Company:** leave empty

Click *Save* to store the peering contract.

4.4.2 Creating Peering Groups

In *System Administration→SIP Peerings*, create a new peering group in the section *Create Peering Group*. You will usually have one peering group per carrier you're planning to send traffic to and receive traffic from. We will create a test group using the peering contract added before:

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

Then click *add* to create the group.

4.4.3 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 the *Name* of your created group in the section *SIP Peering Groups*.

Then add your first peering server in the section *Peering Servers*. 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`
- **Weight:** `1`

Then click *add* 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 domain part in the Request URI. Some peers may require you to set a particular hostname instead of the IP address there, which can be done by filling in this field. The IP address must always be given though, and the request will always be sent to the IP address, no matter what you put into the *hostname* field.

You will now see an additional section *Peering Rules* after your list of peering servers. There you have to define which numbers to route via this peering group.

Important
If you do not add at least one peering rule to your group, the servers in this group will NOT be used for outbound calls. The NGCP will however allow inbound calls from the servers in this group even without peering rules.

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
- **Caller Pattern:** leave empty

- **Description:** `Default Rule`

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

Tip
Enter all phone numbers in full E.164 format, that is *<cc><ac><sn>*.

Important
The selection of peering servers for outbound calls is done in the following order: **1.** length of the matching peering rules for a call. **2.** priority of the peering group. **3.** weight of the peering servers in the selected peering group. After one or more peering group(s) is matched for an outbound call, all servers in this group are tried, according to their weight (lower weight has more precedence). 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 used as a fallback, one after the other until the call succeeds. If no more servers are left in the current peering group, the next group which matches the peering rules is going to be used.

4.4.4 Authenticating and Registering against Peering Servers

Proxy-Authentication for outbound calls

If a peering server requires the SPCE 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* tab of your peer host. To do so, click *Edit* and for example provide the following values:

- **peer_auth_user:** `<username for peer auth>`
- **peer_auth_pass:** `<password for peer auth>`
- **peer_auth_realm:** `<domain for peer auth>`

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>` 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.

Registering at a Peering Server

Unfortunately, the credentials configured above are not yet automatically used to register the SPCE 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/sems/etc/reg_agent.conf.tt2*, then execute *ngcpcfg apply*.

Important
Be aware that this will force SEMS to restart, which will drop running conference calls.

4.5 Configuring Rewrite Rule Sets

Important
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.

You would normally begin with creating Rewrite Rule Sets for the caller and the callee for every of 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.

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.

4.5.1 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.

STRIP LEADING `00` OR +

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

REPLACE `0` BY CALLER'S COUNTRY CODE:

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

NORMALIZE LOCAL CALLS:

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

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, drag&drop the rules into the appropriate order.

4.5.2 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`

REPLACE `0` BY CALLER'S COUNTRY CODE:

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

NORMALIZE AREACODE-INTERNAL CALLS:

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

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.

4.5.3 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`

PREFIX `00` FOR INTERNATIONAL CALLER

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

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

4.5.4 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`

4.5.5 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 4.5 (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.

5 Customer Self-Care Interfaces

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.

5.1 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://<ce-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.

5.1.1 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.

Important
You can simplify the login procedure for one SIP domain in such a way that users in this domain only need to pass the user part (e.g. `user1`) as a username instead of the full web username to log in by setting the parameter *www_csc→site_domain* in the config file */etc/ngcp-config/config.yml* to the corresponding domain (e.g. `1.2.3.4`) and execute `ngcpcfg apply`.

5.1.2 Site Customization

As an operator, you can change the appearance of the CSC panel by modifying a couple of parameters in the section *www_csc→site_confi*
of the config file */etc/ngcp-config/config.yml*. Modify the site title, your company details and the logo to reflect your use case.

You can also enable/disable specific languages a user can choose from in the CSC panel. Currently, English (`en`) and Spanish (`es`) are supported and are activated by default.

After changing one or more of the parameters in this file, execute `ngcpcfg apply` to activate the changes.

5.2 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 4.5.

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 `*50101000`, which then can be used by dialing `*1`.
- **55** - set *One-Shot Reminder Call* e.g. to 08:30 by dialing `*55*0830`.

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`.

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.

5.3 The Voicemail Interface

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`.

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 4.5.) 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.

6 Billing Configuration

This chapter describes the steps necessary to rate calls and export rated CDRs (call detail records) to external systems.

6.1 Billing Data Import

Service billing on the NGCP is based on billing profiles, which may be assigned to VoIP accounts 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.)

6.1.1 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 *System Administration→Billing* and click on *create new billing profile*. You will be taken to a web form where you may enter the following parameters (all values except *handle* and *name* may be left empty):

- **handle**: A unique, permanently fixed string which is used to attach the billing profile to a VoIP account 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.
- **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.

6.1.2 Creating Billing Fees

To set up billing fees, go to *System Administration→Billing* and select *edit fees* next to 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 selecting *create new entry* just below *Stored Billing Fees*. 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`. For input via the web interface, just fill in the text fields accordingly. In both cases, the following information may be specified independently for every destination:

- **destination**: The destination E.164 prefix, SIP domain (or IP address) or SIP URI. May be a simple string (e.g. `431`, `sip.sipwise.com` or `someone@sip.sipwise.com`) or a regular expression matching the complete E.164 number, SIP domain or SIP URI (e.g. `^431.*$`, `^.*@sip\.sipwise\.com$` or `^someone@sip\.sipwise\.com$`). Regular expressions will be stored unmodified, plain strings will be extended exactly as shown in the two examples. The web interface will remove the regular expression prefix and suffix from an entry in the list of store billing fees.

Important
The destination needs to be unique for a billing profile. The system will return an error if a destination 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!**

- **zone**: A zone name for a group of destinations. May be used to group destinations for simplified display, e.g. on invoices. (e.g. `foreign zone 1`)
- **zone detail**: A zone name for a more detailed group of destinations. May be used to group destinations for simplified display, e.g. on invoices. (e.g. `germany landline`)

- **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.

6.1.3 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 *System Administration→Billing* and select *edit peak times* next to 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 *save* to store the setting in the database. You may create more than one off-peak period per weekday, and you may edit existing entries using any of the input fields and clicking *save* next to it. To completely delete a range, just select *delete* next to the entry.

To specify off-peak ranges based on calendar dates, click on *add new date* just below *Dates*. Enter a date in the form of *YYYY-MM-DD* into the *date* input field and fill in the *start* and *end* timestamps as outlined above. Select *save* to store the entry, or *cancel* to close the input form. Existing dates will be listed below, grouped by year. Click on any of the years to view all dates which have been recorded for it. If an entry is added, the corresponding year is expanded automatically. If an already existing date is added, it will overwrite the existing entry.

6.2 Billing Data Export

Regular billing data export is done using CSV (*comma separated values*) files which may be downloaded from the platform using the *cdrexport* user which has been created during the installation.

6.2.1 File Name Format

In order to be able to easily identify billing files, the file name is 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 1: CDR 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. Always `003`.
`<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`.

A valid example filename for a billing file created at 2011-11-10 12:30:00 and being the fourth file exported by the system, is:

```
sipwise_003_20111110123000_0000000004.cdr
```

6.2.2 File Format

Each billing file consists of three parts: one header line, zero to 5000 body lines and one trailer line.

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 2: CDR export file header line format

Body Element	Length	Type	Description
`<version>`	3	zero-padded uint	The format version. Always `003`.
`<number of records>`	4	zero-padded uint	The number of body lines contained in the file.

A valid example for a Header is:

```
003,0738
```

File Body Format

The body consists of a minimum of zero and a maximum of 5000 lines. Each line holds one call detail record in CSV format and is constructed by the following fields, all of them enclosed in single quotes:

Table 3: CDR export file body line format

Body Element	**Length**	**Type**	**Description**
`<id>`	1-10	uint	Internal CDR id.
`<update_time>`	19	timestamp	Timestamp of last modification.
`<source_user_id>`	36	string	Internal UUID of calling party.
`<source_provider_id>`	1-255	string	Internal ID of calling party provider.
`<source_ext_subscriber_id>`	0-255	string	External ID of calling party subscriber.
`<source_ext_contract_id>`	0-255	string	External ID of calling party contract.
`<source_account_id>`	1-10	uint	Interal ID of calling party VoIP account.
`<source_user>`	1-255	string	SIP username of calling party.
`<source_domain>`	1-255	string	SIP domain of calling party.
`<source_cli>`	1-64	string	CLI of calling party in E.164 format.
`<source_clir>`	1	uint	`1` for calls with CLIR, `0` otherwise
`<destination_user_id>`	1 / 36	string	Internal UUID of called party or `0` if callee is not local
`<destination_provider_id>`	1-255	string	Internal ID of called party provider.
`<dest_ext_subscriber_id>`	0-255	string	External ID of called party subscriber.
`<dest_ext_contract_id>`	0-255	string	External ID of called party contract.
`<destination_account_id>`	1-10	uint	Interal ID of called party VoIP account.
`<destination_user>`	1-255	string	Final SIP username of called party.
`<destination_domain>`	1-255	string	Final SIP domain of called party.
`<destination_user_in>`	1-255	string	Incoming SIP username of called party.
`<destination_domain_in>`	1-255	string	Incoming SIP domain of called party.
`<peer_auth_user>`	0-255	string	User to authenticate towards peer.
`<peer_auth_realm>`	0-255	string	Realm to authenticate towards peer.
`<call_type>`	3-4	string	The type of the call - one of: `call`: normal call `cfu`: call forward unconditional `cft`: call forward timeout `cfb`: call forward busy `cfna`: call forward no answer
`<call_status>`	2-7	string	The final call status - one of: `ok`: successful call `busy`: callee busy `noanswer`: no answer from callee `cancel`: cancel from caller `offline` callee offline `timeout`: no reply from callee `other`: unspecified, see `<call_code>` for details
`<call_code>`	3	uint	The final SIP status code.
`<start_time>`	23	timestamp	Timestamp of call start. Seconds include fractional part (3 decimals).
`<duration>`	4-11	fixed precision	Length of call in seconds with 3 decimals.
`<call_id>`	1-255	string	The SIP call-id.
`<rating_status>`	2-7	string	The internal rating status - 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	timestamp	Timestamp of rating or empty if not rated.
`<carrier_cost>`	4-11	fixed precision	The carrier termination cost or empty if not rated. In cent with two decimals.
`<customer_cost>`	4-11	fixed precision	The customer cost or empty if not rated. In cent with two decimals.
`<carrier_zone>`	0-127	string	The carrier billing zone or empty if not rated.

Table 3: (continued)

Body Element	Length	Type	Description
`<customer_zone>`	0-127	string	The customer billing zone or empty if not rated.
`<carrier_destination>`	0-127	string	The carrier billing destination or empty if not rated.
`<customer_destination>`	0-127	string	The customer billing destination or empty if not rated.
`<dialed_digits>`	1-255	string	The user-part of the SIP Request URI as received by the soft-switch.
`<reseller_cost>`	4-11	fixed precision	Currently unused.
`<carrier_free_time>`	1-10	uint	The number of free time seconds used on carrier side or empty if not rated.
`<reseller_free_time>`	1-10	uint	Currently unused.
`<customer_free_time>`	1-10	uint	The number of free time seconds used from the customer's account balance or empty if not rated.
`<reseller_zone>`	0-127	string	Currently unused.
`<reseller_destination>`	0-127	string	Currently unused.
`<line_terminator>`	1	string	A fixed character. Always `\n` (special char LF - ASCII `0x0A`)

A valid example of one body line of a rated CDR is (line breaks added for clarity):

```
  ’2055’,’2007-11-07 11:36:49’,’6b6977f9-6125-4339-a82c-3c5af04652d2’,’1’,,,’1’,’test’,
’sipwise.com’,’43720456700’,’0’,’0’,’3’,,,’0’,’4315551234’,’192.168.101.17’,’4315551234’,
’sipwise.com’,,,’call’,’ok’,’200’,’2007-11-05 16:17:37.641’,’74.731’,
’7F2A3EA1-472F34108EE84@192.168.101.11’,’ok’,’2007-11-07 11:36:49’,’9.25’,’16.03’,
’national landline’,’national landline’,’landline vienna’,’landline vienna’,’015551234’,
’0.00’,’0’,’0’,’0’,’’,’’
```

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
```

6.2.3 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 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_keys` 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 `cdrexport` user is kept in a jailed environment on the system, so it has only access to a very limited set of commandline utilities.

7 Provisioning interfaces

The sip:provider CE provides two provisioning interfaces for easy interconnection with 3rd party tools. The user can access all the functionalities provided by the Admin interface or the CSC interface via SOAP or XMLRPC interfaces. The server provides online documentation about all the functions available. To access the online documentation for the first time, you need to follow the following instructions:

- Generate a password for http access to the provisioning interfaces:

```
htpasswd -nbs myuser mypassword
```

Note
Also see `man 1 htpasswd` on how to generate crypt or MD5 passwords if you like. Of course you may use any other process to generate crypt, MD5 or SHA hashed passwords. But using `htpasswd` ensures the hashes are also understood by Apache.

- Edit */etc/ngcp-config/config.yml*. Under section *ossbss→htpasswd*, replace *user* and *pass* with your new values and execute *ngcpcfg apply* as usual.
- Access https://<ce-ip>:2443/SOAP/Provisioning.wsdl and login with your new credentials.

Note
The default port for provisioning interfaces is 2443. You can change it in */etc/ngcp-config/config.yml* by modifying *ossbss→apache→port* and execute *ngcpcfg apply*.

Important
The displayed online API documentation shows all the currently available functionalities. Enabling or disabling features in */etc/ngcp-config/config.yml* will directly reflect in the functions being available via the APIs.

8 Configuration Framework

The sip:provider CE 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* 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 6 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 and constants.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.

8.1 Configuration templates

The sip:provider CE 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/sems/sems.conf* are stored in */etc/ngcp-config/templates/etc/sems/*.

There are different types of files in this template framework, which are described below.

8.1.1 .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:provider CE system. The configuration framework will combine these files with the values provided by *config.yml* and *constants.yml* to generate the appropriate configuration file.

Example: In the installation chapter we've changed the parameter *networking* → *eaddress* from the default *127.0.0.1* to our public IP address *1.2.3.4*. This parameter will for example change kamailio's listen address, when the configuration file is generated. A quick look to the template file under */etc/ngcp-config/templates/etc/kamailio/lb/kamailio.cfg.tt2* will show a line like this:

```
listen=udp:[% networking.eaddress %]:[% kamailio.lb.port %]
```

After applying the changes with the *ngcpcfg apply* command, a new configuration file will be created under */etc/kamailio/kamailio.cfg* with the proper values taken from the main configuration file:

```
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. Anyways, 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/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
ngcpcfg apply
```

The ngcpcfg 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 ngcpcfg's template files (all the ones with the .tt2 suffix).

8.1.2 .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
```

8.1.3 .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* is associated to */etc/asterisk/sip.conf*
- The general one, named *ngcpcfg.services* wich is associated to every file in its target directory. Example: */etc/ngcp-config/templates/etc* 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.

8.2 config.yml and constants.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/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.

8.3 ngcpcfg and its command line options

The ngcpcfg utility supports the following command line options:

8.3.1 apply

The *apply* option is a short-cut for the options "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.

8.3.2 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/apache2/sites-available/ngcp-www-admin you can execute:

```
ngcpcfg build /etc/apache2/sites-available/ngcp-www-admin
```

Example: to generate all the files located inside the directory /etc/apache2/ you can execute:

```
ngcpcfg build /etc/apache2/
```

8.3.3 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.

8.3.4 decrypt

Decrypt /etc/ngcp-config-crypted.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.

8.3.5 diff

Show uncommited changes between ngcpcfg's Git repository and the working tree inside */etc/ngcp-config*. Iff the tool doesn't report anything it means that there are no uncommited 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.

8.3.6 encrypt

Encrypt /etc/ngcp-config and all resulting configuration files with a user defined password and save the result as /etc/ngcp-config-crypted.tgz.gpg. Note: This feature is only available if the ngcp-ngcpcfg-locker package is installed.

8.3.7 help

The *help* options displays ngcpcfg's help screen and then exits without any further actions.

8.3.8 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.

8.3.9 pull

Retrieve modifications from shared storage. Note: This option is available in the High Availability setup only.

8.3.10 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. If no host has 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.

8.3.11 services

The *services* option executes the service handlers for any modified configuration file(s)/directory.

8.3.12 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.

9 Security and Maintenance

Once the sip:provider CE 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.

9.1 Firewalling

The sip:provider CE runs a wide range of services. Some of them need to interact with the user, while some others need to interact with the administrator or with nobody at all. Assuming that we trust the sip:provider CE server for outgoing connections, we'll focus only on incoming traffic to define the services that need to be open for interaction.

Table 4: Subscribers

Service	**Default port**	**Config option**
Customer self care interface	443 TCP	www_csc→apache→port
SIP	5060 UDP, TCP	kamailio→lb→port
SIP over TLS	5061 TCP	kamailio→lb→tls→port + kamailio→lb→tls→enable (Disabled by default)
RTP	30000-40000 UDP	rtpproxy→minport + rtpproxy→maxport

Table 5: Administrators

Service	Default port	Config option
SSH/SFTP	22 TCP	NA
Administrator interface	1443 TCP	www_admin→apache→port
Provisioning interfaces	2443 TCP	ossbss→apache→port

Caution
To function correctly, the *mediaproxy* requires an additional *iptables* rule installed. This rule (with a target of `MEDIAPROXY`) is automatically installed and removed when the mediaproxy 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 `MEDIAPROXY` 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 MEDIAPROXY --id 0`

9.2 Password management

The sip:provider CE 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 document.

- The default password of the system account *cdrexport* is *cdrexport*. 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 generated.
- The *root* user in MySQL has no default password. A password should be set using the *mysqladmin password* command.
- The administrative web interface has a default user *administrator* with password *administrator*. It should be changed within this interface.

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.

9.3 SSL certificates.

The sip:provider CE 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→sslcertfile* and *ossbss→apache→sslcertkeyfile* for the *provisioning interface*.
  - *www_admin→apache→sslcertfile* and *www_admin→apache→sslcertkeyfile* for the *admin interface*.

- *www_csc→apache→sslcertfile* and *www_csc→apache→sslcertkeyfile* for the *customer self care interface*.

- Apply the configuration changes with *ngcpcfg apply*.

The sip:provider CE 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→sslcertfile* and *kamailio→lb→tls→sslcertkeyfile* .
- Apply the configuration changes with *ngcpcfg apply*.

9.4 Backup and recovery

9.4.1 Backup

The sip:provider CE can be integrated with most of the existing backup solutions. While it does not provide any backup system by default, any Debian compatible system can be installed. It's not the scope of this chapter to go through backup system configuration. We'll focus on which information needs to be saved.

The minimum set of information to be backed up is:

- The database information.

This is the most important data in the system. All subscriber information, billing, CDRs, user preferences etc. are stored in the MySQL server. A periodical dump of all the databases should be performed.

- System configuration options

/etc/ngcp-config/config.yml, */etc/ngcp-config/constants.yml*, */etc/mysql/debian.cnf* and */etc/mysql/sipwise.cnf* files, where your specific system configurations are stored, should be included in the backup as well.

- Optional: Exported CDRs

The directory */home/jail/home/cdrexport* contains the exported CDRs the system has generated so far. It depends on your local call data retention policy whether or not to remove these files after exporting them to an external system.

- Optional: Custom files

Any custom configurations, like modified templates or additionally implemented services which are not provided by the sip:provider CE

9.4.2 Recovery

In the worst case scenario, when the system needs to be recovered from a total loss, you only need 4 steps to get back online:

- Install the sip:provider CE as explained in chapter 2.
- Restore *config.yml*, *constants.yml*, *debian.cnf* and *sipwise.cnf* from the backup, overwriting your local files.
- Restore the database dump.
- Execute *ngcpcfg apply*.

9.5 Reset database

To reset database to its original state you can use the script provided by CE: * Execute *ngcp-reset-db*. It will assign new unique password for the NGCP services and restart all services. IMPORTANT: All existing data will be wiped out without possibility of restoring.