OpenH323 Gatekeeper - The GNU Gatekeeper

Maintainer of this manual: *Chih-Wei Huang <cwhuang@linux.org.tw>*

v2.0.3, 20 February 2003

This is the User Manual for how to compile, install, configure and monitor OpenH323 Gatekeeper - The GNU Gatekeeper.

Contents

1 Introduction

1.1 About

OpenH323 Gatekeeper - The GNU Gatekeeper <http://www.gnugk.org/> is an open-source project that implements an H.323 gatekeeper. A gatekeeper provides call control services to the H.323 endpoints. It is an integral part of most usefull internet telephony installations that are based on the H.323 standard.

According to Recommendation H.323, a gatekeeper shall provide the following services:

- Address Translation
- Admissions Control
- Bandwidth Control
- Zone Management
- Call Control Signaling
- Call Authorization
- Bandwidth Management
- Call Management

The GNU Gatekeeper implements most of these functions based on the *OpenH323* <http://www.openh323.org/> protocol stack.

Recommendation H.323 is an international standard published by the *ITU* <http://www.itu.int/>. It is a communications standard for audio, video, and data over the Internet. See also Paul Jones' *primer on H.323* <http://www.packetizer.com/iptel/h323/>.

For a detailed description of what a gatekeeper does, see *here* <http://www.webproforum.com/h323/topic06.html>.

1.2 Copyright

It is covered by the *GNU General Public License* (GNU GPL). In addition to that, we explicitely grant the right to link this code to the OpenH323 and OpenSSL library.

Generally speaking, the GNU GPL allows you to copy, distribute, resell or modify the softwares, but it requires that all derived works must be published under GNU GPL also. That means that you must publish full source for all extensions to the gatekeeper and for all programs you include the gatekeeper into. See the file COPYING for details.

If that's not what you want, you must interface to the gatekeeper through the status thread and communicatie via TCP with it. That way you only have to integrate the basic funtionality into the gatekeeper (and provide source for that) and can keep other parts of your application private.

1.3 Name

The formal name of this project is *OpenH323 Gatekeeper - The GNU Gatekeeper*, shortly *OpenH323GK* or *GnuGK*. Please don't confuse it with other gatekeeper projects.

There are several open-source gatekeeper projects based on the OpenH323 protocol stack.

- *OpenGatekeeper* `<http://opengatekeeper.sourceforge.net/>` - by *Egoboo* `<http://www.egoboo.com/>`

  A full featured gatekeeper freely available under MPL. The project has been inactive for a period of time now. There is an H.323 proxy based on OpenGatekeeper, see *OpenH323Proxy* `<http://openh323proxy.sourceforge.net/>`.

- *OpenGK* `<http://www.openh323.org/code.html>` - by *Equivalence* `<http://www.equival.com.au/>`

  Only in a very primary grades.

- OpenH323 Gatekeeper - this one.

To have different gatekeepers with very similar names is really confusing for most users. Since our "OpenH323 Gatekeeper" was the first on the scene, it is not our fault that others have chosen similar names. But to make the destinction a little more clear without confusing people even more, we have decided to give the project a subtitle "OpenH323 Gatekeeper - The GNU Gatekeeper" and start using gnugk as name for executables.

1.4 Features

The version 2.0.3 is a bugfix release, plus a little enhancement:

- Forward a call Setup to the specified endpoint directly on receiving Q.931 Facility with reason **callForwarded**.
- Allow specify NATed endpoints manually.
- Added a simple form of inbound call distribution. Calls to a VirtualQueue can be routed to agents by an external distribution application.

The new features added to version 2.0.2 are:

- Add Citron's NAT Technology that allows transparently penetrate NAT boxes.Support multiple endpoints and calls concurrently.
- The gatekeeper can sit behind an NAT box and registered by endpoints with public IPs.
- New extended fd_set structure, which allow the gatekeeper to support thousands of concurrent calls in routed mode.
- Support QoS by adding TOS flag to RTP/RTCP packets.
- Login status port by username and password.

Of course, the major functions in version 2.0 are also included:

- The registration table and call record table are redesigned, thread-safe, and very efficient. Support ten thousands of registrations and thousands of concurrent calls.
- A new routed mode architecture that support H.225.0/Q.931 routed and H.245 routed without forking additional threads. Thus the thread number limit will not restrict the number of concurrent calls.
- Support H.323 proxy by routing all logical channels, including RTP/RTCP media channels and T.120 data channels. Logical channels opened by H.245 tunnelling and fast-connect procedure are also supported. In proxy mode, there is no traffic between the calling and called parties directly. Thus it is very useful if you have some endpoints using private IP behind an NAT box and some endpoints using public IP outside the box.

- Support gatekeepers cluster by exchanging LRQ/LCF/LRJ (neighboring function). If the destination of a received LRQ is unknown, the GnuGK can forward it to next hop. Therefore the GnuGK can work as a directory gatekeeper.
- Support various authentication methods for selectable RAS requests, including H.235 password (MD5 & SHA-1), IP pattern and prefixes matching. MySQL and LDAP are supported as backend database for authentication.
- Support alternate gatekeepers for redundancy and load balancing. If the GnuGK is overloaded, the endpoints can be redirected to other gatekeepers.
- Can work as an endpoint (gateway or terminal) by resigtering with a parent gatekeeper. With this feature, building gatekeeper hierarchies is easily.
- Monitor and control the GnuGK via TCP status port, including registration and call statistics.
- Output CDR(call detail record) to status port for backend billing system. The CDR contains call identifier, calling and called IP, start and end time and call duration.
- Most configurations are changeable at runtime. The GnuGK rereads the configurations on receiving '`reload`' command via status port, or on receiving `HUP` signal (Unix platform).

1.5 Download

The newest stable and a development version are available at *the download page* `<http://www.gnugk.org/h323download.html>`.

The very latest source code is in the CVS at *Sourceforge* `<http://sourceforge.net/cvs/?group_id=4797>` (*Web-GUI* `<http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/openh323gk/>`). Beware - that's the bleeding edge.

You can also download executables from *Sourceforge* `<http://sourceforge.net/project/showfiles.php?group_id=4797>`. Only some versions are made available as executables.

1.6 Mailing Lists

There are two mailing list for the project, one for the developers and one for the users.

General user questions should be send to the *users mailing list* `<mailto:Openh323gk-users@sourceforge.net>`. You can find the list archive *here* `<http://sourceforge.net/mailarchive/forum.php?forum_id=8549>`. To join this mailing list, click *here* `<http://lists.sourceforge.net/lists/listinfo/openh323gk-users>`.

To report problems or submit bugs/patches, send mails to the *developers mailing list* `<mailto:Openh323gk-developer@sourceforge.net>`. The list archive is *here* `<http://sourceforge.net/mailarchive/forum.php?forum_id=3079>`. Please send user questions to the users mailinglist and keep this list to development! If you want to contribute to the project, please *join the mailing list* `<http://lists.sourceforge.net/lists/listinfo/openh323gk-developer>`.

Note: Please don't send your questions as private emails to individual developer. We are usually busy. We would **not** like to be your private consultant, unless you'd like to pay us. Send your problems to the appropriate public mailing list so everybody can help you.

Please also note don't send the GnuGK specific problems to the OpenH323 mailing list, or vice versa. Or you will definitely be ignored. They are different projects, though closely related. The developers are also different, though they usually work together in some ways.

Before you sending an email, make sure you have read the related documents carefully. Describe your problems clearly and precisely. Show us the error messages or logs if there is any. If you don't know how to ask questions in a public forum, read Eric S. Raymond's famous article *How To Ask Questions The Smart Way* `<http://www.tuxedo.org/~esr/faqs/smart-questions.html>` before you asking.

1.7 Contributors

The current project coordinator is *Jan Willamowius* `<http://www.willamowius.de/>` <*jan@willamowius.de*>

The main features and functions of version 2.0 are contributed by *Chih-Wei Huang* `<http://www.cwhuang.idv.tw/>` <*cwhuang@linux.org.tw*> and *Citron Network Inc.* `<http://www.citron.com.tw/>`, including thread-safe registration and call tables, new routed mode architecture, H.323 proxy, H.235 authentication and MySQL backend.

A team at *mediaWays* `<http://www.mediaways.net/>` is working on LDAP database-subsystem, overlapped sending and advanced routing mechanisms.

The initial version of the gatekeeper has been developed by Xiang Ping Chen, Joe Metzger and Rajat Todi.

2 Compiling and Installing

2.1 Compiling the Gatekeeper

To build the gatekeeper you need at least PWLib 1.2 and OpenH323 1.8 or later. The development version of the gatekeeper usually needs the most recent OpenH323 version available. These libraries are available at *OpenH323 Download Page* `<http://www.openh323.org/code.html>`. See the instructions on *how to compile the OpenH323 code* `<http://www.openh323.org/build.html>`.

Order of compiling:

1. PWLib (release + debug version)
2. OpenH323
3. OpenH323 test application (not needed, just to make sure everything works so far)
4. The Gatekeeper

On Unix do a `make debug` or `make opt` in the gatekeeper directory to build debug or release version, respectively. Use `make both` to build both versions. Note you have to use GCC 2.95.2 or later. The older version may not work.

On Windows just open and compile the provided project (`gk.dsw`) for Microsoft Visual C++ 6.0 or 7.0 (Visual C++ 5.0 is too old).

Since version 2.0, the Gatekeeper supports MySQL and LDAP backend databases. If you don't want MySQL support, you may set NO_MYSQL environment before making:

```
$ NO_MYSQL=1 make both
```

To leave out LDAP support:

```
$ NO_LDAP=1 make both
```

Or disable both with

```
$ NO_MYSQL=1 NO_LDAP=1 make both
```

Since version 2.0.1 the Gatekeeper has implemented an extended fd_set structure that enables the Gatekeeper to support thousands of concurrent calls in routed mode. To enable this feature, export LARGE_FDSET environment variable to the maximum number of file descriptors. For example,

```
$ LARGE_FDSET=16384 make opt
```

2.2 Installing the Gatekeeper

There is no special installation procedure needed. Just copy the executable to the directory you like and create a config file for it. There are several config examples in the `etc/` subdirectory of source tree. See section 4.2 (Configuration File) for detailed explanations.

For example, in Linux x86 platform, the optimized executable `gnugk` is produced in `obj_linux_x86_r/` subdirectory. You may copy it to `/usr/sbin/`, create a config in `/etc/gnugk.ini` and start it by

```
$ /usr/sbin/gnugk -c /etc/gnugk.ini -o /var/log/gnugk.log -ttt
```

See section 4.1 (Command Line Options) for details.

2.3 Pre-Built Binaries

If you do not wish to compile the gatekeeper from source, there are several pre-built 'packages' available from *SourceForge* `<http://sourceforge.net/project/showfiles.php?group_id=4797>`. Not all versions will be made available as binaries - check what is avilable.

Red Hat packages (.rpm)

Download the RPMs and enter the following command as `root`, substitute in the name of the file you downloaded.

```
$ rpm -Uvh gnugk-x.x.x.rpm
```

Debian packages (.deb)

If you are using the 'stable' (woody) branch of Debian, you can install the gatekeeper by using the following command as `root`:

```
$ apt-get install openh323gk
```

3 Getting Started (Tutorial)

3.1 A first simple experiment

To see that all components are up and running, get 2 Linux workstations, both connected to the LAN. Make sure you have at least version 1.1 of OpenH323 and OhPhone installed. On the first machine run the gatekeeper and ohphone (on different consoles):

```
jan@machine1 > gnugk -ttt
```

Now the gatekeeper is running in direct mode. The "-ttt" option tells the gatekeeper to do a lot of debug output on the console (you can direct that output to a file with "-o logfile").

```
jan@machine1 > ohphone -l -a -u jan
```

Now this OhPhone is listening (-l) for calls and will automatically accept them (-a). It has registered as user jan with the gatekeeper that it will automatically detect. (If the auto detect fails for some reason use "-g 1.2.3.4" to specify the IP number the gatekeeper is running on.)

On the second machine run ohphone only:

```
peter@machine2 > ohphone -u peter jan
```

The second instance of OhPhone registers with the auto detected gatekeeper as user peter and tries to call user jan. The gatekeeper will resolve the username to the IP number from where user jan has registered (machine1 in this case) and OhPhone will call the other instance of OhPhone on machine one.

The first instance of OhPhone will accept that call and Peter and Jan can chat.

3.2 Using the Status interface to spy on the gatekeeper

Now we try to see which messages are handled by the gatekeeper. On a new console on machine1 we use telnet to connect to the gatekeeper:

```
jan@machine1 > telnet machine1 7000
```

Most probably we'll get an "Access forbidden!" message, because not everybody is allowed to spy.

Now we create a file called gatekeeper.ini and put it in the directory where we start the gatekeeper. gatekeeper.ini only contains 4 lines:

```
[Gatekeeper::Main]
Fourtytwo=42
[GkStatus::Auth]
rule=allow
```

Stop the gatekeeper with Ctrl-C and restart it. When we do the telnet again, we stay connected with the gatekeeper. Now repeat the first experiment where Peter calls Jan and see which messages are handled by the gatekeeper in non-routed mode. There is a number of commands that can be issued in this telnet session: Type "help" to see them. To end the telnet session with the gatekeeper type "quit" and hit Enter.

3.3 Starting the gatekeeper in routed mode

Starting the gatekeeper in routed mode means that the gatekeeper uses "gatekeeper routed signalling" for all calls. In this mode the gatekeeper all signalling messages go through the gatekeeper and it has much greater control over the calls.

```
jan@machine1 > gnugk -r
```

Now the gatekeeper is running in routed mode. Telnet to the status port and make a call to see what messages are now handled by the gatekeeper.

Note that all media packets (audio and video) are still sent directly between the endpoints (the 2 instances of ohphone).

Since gatekeeper routed signalling is much more complicated you are much more likely to hit a bug n the gatekeeper in this mode. But if it breaks, you get to keep the pieces. ;-)

3.4 A virtual PBX: Disconnecting calls

Until now the gatekeeper has only acted as a mechanism to resolve symbolic names to IP addresses. Thats an important function but hardly exciting.

Since the gatekeeper has a lot of control over the calls, it can terminate them for example. When we are connected to the status port, we can list all active calls with "`PrintCurrentCalls`". To terminate a call, we can say "`Disconnectip 1.2.3.4`" for one of its endpoints.

One could for example write a simple script that connects to the status port and listens for all ongoing calls and terminates them after 5 minutes, so no user can over use system resources.

This mechanism could also be used to call transfers between users or call forwarding. (But thats not yet implemented in the gatekeeper.)

3.5 Routing calls over a gateway to reach external users

Without using a gateway you can only call other people with an IP phone over the Internet. To reach people with ordinary telephones you must use a gateway.

```
-----------------          --------------
| endpoint "jan"|          |            |
| 192.168.88.35 |--------->| Gatekeeper |
|_______________|          |            |
-----------------          |            |
| gateway "gw1" | outgoing |            |
| 192.168.88.37 |<---------|____________|
|_______________|
```

The gatekeeper has to know which calls are supposed to be routed over the gateway and what numbers shall be called directly. Use the [RasSrv::GWPrefixes] section of the config file to tell the gatekeeper the prefix of numbers that shall be routed over the gateway.

```
[RasSrv::GWPrefixes]
gw1=0
```

This entry tells the gatekeeper to route all calls to E.164 numbers starting with 0 to the gateway that has registered with the H.323 alias "gw1". If there is no registered gateway with that alias the call will fail. (Note that you must use the gateway alias - you can't just tell the gatekeeper the IP number of the gateway.)

3.6 Rewriting E.164 numbers

When using a gateway you often have to use different numbers internally and rewrite them before sending them over a gateway into the telephone network. You can use the 4.2.6 (RasSrv::RewriteE164) section to configure that.

Example: You want to call number 12345 with you IP Phone and would like to reach number 08765 behind a gateway called "gw1".

```
[RasSrv::GWPrefixes]
gw1=0

[RasSrv::RewriteE164]
12345=08765
```

4 Using the Gatekeeper (Reference)

The behavior of the gatekeeper is completely determined by the command line options and configuration file. Some command line options may override the setting of the configuration file. For example, the option `-l` overrides the setting `TimeToLive` in the configuration file.

4.1 Command Line Options

Almost every option has a short and a long format, e.g., `-c` is the same as `-config`.

4.1.1 Basic

`-h -help`

Show all available options and quit the program.

`-c -config filename`

Specify the configuration file to use.

`-s -section section`

Specify which main section to use in the configuration file. The default is [Gatekeeper::Main].

`-i -interface IP`

Specify the interface(IP) that the gatekeeper listens to. You should leave out this option to let the gatekeeper automatically determine the IP it listens to, unless you want the gatekeeper only binds to a specified IP.

`-l -timetolive n`

Specify the time-to-live timer in seconds for endpoint registration. It overrides the setting `TimeToLive` in the configuration file. See 4.2.1 (there) for detailed explanations.

`-b -bandwidth n`

Specify the total bandwidth available for the gatekeeper. Without specifying this option, the bandwidth management is disable by default.

`-pid filename`

Specify the pid file, only valid for Unix version.

4.1.2 Gatekeeper Mode

The options in this subsection override the settings in the 4.2.2 ([RoutedMode] section) of the configuration file.

`-d -direct`

Use direct endpoint call signalling.

`-r -routed`

Use gatekeeper routed call signalling.

`-rr -h245routed`

Use gatekeeper routed call signalling and H.245 control channel.

4.1.3 Debug Information

`-o -output filename`

Write trace log to the specified file.

`-t -trace`

Set trace verbosity. The more `-t` you add, the more verbose to output. For example, use `-ttttt` to set the trace level to 5.

4.2 Configuration File

The configuration file is a standard text file. The basic format is:

```
[Section String]
Key Name=Value String
```

Comments are marked with a hash (#) or a semicolon (;) at the beginning of a line.

The file `complete.ini` contains all available sections for the GnuGK. In most cases it doesn't make sense to use them all at once. The file is just meant as a collection of examples for many settings.

The configuration file can be changed at runtime. Once you modify the configuration file, you may issue `reload` command via status port, or send a signal `HUP` to the gatekeeper process on Unix. For example,

```
kill -HUP ‘cat /var/run/gnugk.pid‘
```

Note It is said that some section names in GnuGK 2.0 are [RasSrv::*], while others are [RasSvr::*]. This inconsistency confuses users. In 2.0.1 all sections are corrected to [RasSrv::*]. If you upgrade from 2.0 or eariler version, remember to change the section names, or the GnuGK will refuse to start.

4.2.1 Section [Gatekeeper::Main]

- `Fourtytwo=42`
  Default: `N/A` This setting is used to test the presence of the config file. If it is not found, a warning is issued. Make sure it's in all your config files.

- `Name=OpenH323GK`
  Default: `OpenH323GK` Gatekeeper identifier of this gatekeeper. The gatekeeper will only respond to GRQs for this ID and will use it in a number of messages to its endpoints.

- `Home=192.168.1.1`
  Default: `0.0.0.0` The gatekeeper will listen for requests on this IP number. By default, the gatekeeper listens on all interfaces of your host. You should leave out this option, unless you want the gatekeeper only to bind to a specified IP.

- `NetworkInterfaces=192.168.1.1/24,10.0.0.1/0`
  Default: `N/A` Specify the network interfaces of the gatekeeper. By default the gatekeeper will detect the interfaces of your host automatically. There are two situations that you may want to use this option. One is automatical detection failed, another is the gatekeeper is behind an NAT box and allow endpoints with public IPs to register with. In this case you should set the option just as the gatekeeper is running on the NAT box.

- `EndpointIDSuffix=_gk1`
  Default: `_endp` The gatekeeper will assign a unique identifier to each registered endpoint. This option can be used to specify a suffix to append to the endpoint identifier. This is only usefull when using more than one gatekeeper.

- `TimeToLive=300`
  Default: `-1` An endpoint's registration with a gatekeeper may have a limited life span. The gatekeeper specifies the registration duration of an endpoint by including a **timeToLive** field in the RCF message. After the specified time, the registration has expired. The endpoint shall periodically send an RRQ having the **keepAlive** bit set prior to the expiration time. Such a message may include a minimum amount of information as described in H.225.0. This is called a lightweight RRQ.

  This configuration setting specifies the time-to-live timer in seconds until the registration expires. Note the endpoint may request a shorter **timeToLive** in the RRQ message to the gatekeeper. To avoid an overload of RRQ messages, the gatekeeper automatically adjusts this timer to 60 seconds if you give a lesser value!

  After the expiration time, the gatekeeper will subsequently send two IRQ messages to query if the endpoint is still alive. If the endpoint responds with an IRR, the registration will be extended. Otherwise the gatekeeper will send a URQ with reason **ttlExpired** to the endpoint. The endpoint must then re-register with the gatekeeper using a full RRQ message.

  To disable this feature, set it to `-1`.

- `TotalBandwidth=100000`
  Default: `-1` Total bandwidth available to be given to endpoints.

- `RedirectGK=Endpoints > 100 || Calls > 50`
  Default: `N/A` This option allow you to redirect endpoints to alternate gatekeepers when the gatekeeper overloaded. For example, with the above setting the gatekeeper will reject an RRQ if registered endpoints exceed 100, or reject an ARQ if concurrent calls exceed 50.

  Furthermore, you may explicitly redirects all endpoints by setting this option to `temporary` or `permanent`. The gatekeeper will return an RAS rejection message with a list of alternate gatekeepers defined in `AlternateGKs`. Note that a `permanent` redirection means that the redirected endpoints will not register with this gatekeeper again. Please also note the function only takes effect to H.323 version 4 compliant endpoints.

- `AlternateGKs=1.2.3.4:1719:false:120:OpenH323GK`
  Default: `N/A` We allow for existence of another gatekeeper to provide redundancy. This is implemented in a active-active manner. Actually, you might get into a (valid !) situation where some endpoints are registered with the first and some are registered with the second gatekeeper. You should even be able use the two gatekeepers in a round_robin fashion for load-sharing (that's untested, though :-)). If you read on, "primary GK" refers to the gatekeeper you're currently configuring and "alternate GK" means the other one. The primary GK includes a field in the RCF to tell endpoints which alternate IP and gatekeeper identifier to use. But the alternate GK needs to know about every registration with the primary GK or else it would reject calls. Therefore our gatekeeper can forward every RRQ to an alternate IP address.

The AlternateGKs config option specifies the fields contained in the primary GK's RCF. The first and second fields of this string define where (IP, port) to forward to. The third tells endpoints whether they need to register with the alternate GK before placing calls. They usually don't because we forward their RRQs, so they get registered with the alternate GK, too. The fourth field specified the priority for this GK. Lower is better, usually the primary GK is considered to have priority 1. The last field specifies the alternate gatekeeper's identifier.

- `SendTo=1.2.3.4:1719`
  Default: `N/A` Although this information is contained in AlternateGKs, you must still specify which address to forward RRQs to. This might differ from AlternateGK's address, so it's a separate config option (think of multihomed machines).

- `SkipForwards=1.2.3.4:5.6.7.8`
  Default: `N/A` To avoid circular forwarding, you shouldn't forward RRQs you get from the other GK (this statement is true for both, primary and alternate GK). Two mechanisms are used to identify whether a request should be forwarded. The first one looks for a flag in RRQ. Since few endpoints implement this, we need a second, more reliable way. Specify the other gatekeeper's IP in this list.

- `StatusPort=7000`
  Default: `7000` Status port to monitor the gatekeeper. See 5 (this section) for details.

Most users will never need to change any of the following values. They are mainly used for testing or very sophisticated applications.

- `UseBroadcastListener=0`
  Default: `1` Defines whether to listen to broadcast RAS requests. This requires binding to all interfaces on a machine so if you want to run multiple instances of gatekeepers on the same machine you should turn this off.

- `UnicastRasPort=1719`
  Default: `1719` The RAS channel TSAP identifier for unicast.

- `MulticastPort=1718`
  Default: `1718` The RAS channel TSAP identifier for multicast.

- `MulticastGroup=224.0.1.41`
  Default: `224.0.1.41` The multicast group for the RAS channel.

- `EndpointSignalPort=1720`
  Default: `1720` Default port for call signalling channel of endpoints.

- `ListenQueueLength=1024`
  Default: `1024` Queue length for incoming TCP connection.

- `SignalReadTimeout=1000`
  Default: `1000` Time in miliseconds for read timeout on call signalling channels (Q931).

- `StatusReadTimeout=3000`
  Default: `3000` Time in miliseconds for read timeout on status channel.

- `StatusWriteTimeout=5000`
  Default: `5000` Time in miliseconds for write timeout on status channel.

4.2.2 Section [RoutedMode]

Call signalling messages may be passwd in two ways. The first method is Direct Endpoint Call Signalling, in which case the call signalling messages are passed directly between the endpoints. The second method is Gatekeeper Routed Call Signalling. In this method, the call signalling messages are routed through the gatekeeper between the endpoints. The choice of which methods is used is made by the gatekeeper.

When Gatekeeper Routed call signalling is used, the gatekeeper may choose whether to route the H.245 control channel and logical channels.

Case I.

The gatekeeper doesn't route them. The H.245 control channel and logical channels are established directly between the endpoints.

Case II.

The H.245 control channel is routed between the endpoints through the gatekeeper, while the logical channels are established directly between the endpoints.

Case III.

The gatekeeper routes the H.245 control channel, as well as all logical channels, including RTP/RTCP for audio and video, and T.120 channel for data. In this case, no traffic is passed directly between the endpoints. This is usually called an H.323 Proxy, which can be regarded as an H.323-H.323 gateway.

This section defines the gatekeeper routed mode options (case I & II). The proxy feature is defined in the 4.2.3 (next section). All settings in this section are affected by reloading.

- `GKRouted=1`
  Default: `0` Whether to enable the gatekeeper routed mode.

- `H245Routed=1`
  Default: `0` Whether to route the H.245 control channel. Only takes effect if `GKRouted=1`.

- `CallSignalPort=0`
  Default: `1721` The port of call signalling for the gatekeeper. The default port is `1721`. We don't use the well-known port `1720` so you can run an H.323 endpoint in the same machine of the gatekeeper. You may set it to `0` to let the gatekeeper choose an arbitrary port.

- `CallSignalHandlerNumber=2`
  Default: `1` The number of call signalling handler. You may increase this number in a heavy loaded gatekeeper. The number can only be increased at runtime. If you have a SMP machine, you can set this number to your number of CPUs.

- `AcceptNeighborsCalls=1`
  Default: `1` With this feature enabled, the call signalling thread will accept calls without a pre-existing CallRec found in the CallTable, provided an endpoint corresponding to the destinationAddress in Setup can be found in the RegistrationTable, and the calling party is its neighbors or parent GK. The gatekeeper will also use it's own call signalling address in LCF in responding to an LRQ. That means, the call signalling will be routed to GK2 in GK-GK calls. As a result, the CDRs in GK2 can correctly show the connected time, instead of 'unconnected'.

- `AcceptUnregisteredCalls=1`
  Default: `0` With this feature enabled, the gatekeeper will accept calls from any unregistered endpoint. However, it raises security risks. Be careful to use it.

- `RemoveH245AddressOnTunneling=1`
  Default: `0` Some endpoints send h245Address in the UUIE of Q.931 even when h245Tunneling is set to TRUE. This may cause interoperability problems. If the option is TRUE, the gatekeeper will remove h245Address when h245Tunneling flag is TRUE. This enforces the remote party to stay in tunnelling mode.

- `RemoveCallOnDRQ=0`
  Default: `1` With this option turning off, the gatekeeper will not disconnect a call if it receives a DRQ for it. This avoids potential race conditions when a DRQ overtakes a Release Complete. This is only meaningful in routed mode because in direct mode, the only mechanism to signal end-of-call is a DRQ.

- `DropCallsByReleaseComplete=1`
  Default: `0` According to Recommendation H.323, the gatekeeper could tear down a call by sending RAS DisengageRequest to endpoints. However, some bad endpoints just ignore this command. With this option turning on, the gatekeeper will send Q.931 Release Complete instead of RAS DRQ to both endpoints to force them drop the call.

- `SendReleaseCompleteOnDRQ=1`
  Default: `0` On hangup, the endpoint sends both Release Complete within H.225/Q.931 and DRQ within RAS. It may happen that DRQ is processed first, causing the gatekeeper to close the call signalling channel, thus preventing the Release Complete from being forwarding to the other endpoint. Though the gatekeeper closes the TCP channel to the destination, some endpoints (e.g. Cisco CallManager) don't drop the call even if the call signalling channel is closed. This results in phones that keep ringing if the caller hangs up before the callee pickups. Setting this parameter to `1` makes the gatekeeper always send Release Complete to both endpoints before closing the call when it receives DRQ from one of the parties.

- `SupportNATedEndpoints=1`
  Default: `0` Whether to allow an endpoint behind an NAT box register to the gatekeeper. If yes, the gatekeeper will translate the IP address in Q.931 and H.245 channel into the IP of NAT box.

  Since 2.0.2, the GnuGK supports NAT outbound calls (from an endpoint behind NAT to public networks) directly without any necessary modification of endpoints or NAT box. Just register the endpoint with the GnuGK and you can make call now.

- `ScreenDisplayIE=MyID`
  Default: `N/A` Modify the DisplayIE of Q.931 to the specified value.

- `ScreenCallingPartyNumberIE=0965123456`
  Default: `N/A` Modify the CallingPartyNumberIE of Q.931 to the specified value.

- `ForwardOnFacility=1`
  Default: `1` If yes, on receiving Q.931 Facility with reason **callForwarded**, the gatekeeper will forwards call Setup directly to the forwarded endpoint, instead of passing the message back to the caller. If you have broken endpoints that can't handle Q.931 Facility with reason **callForwarded**, turn on this option.

- `ShowForwarderNumber=0`
  Default: `0` Whether to rewrite the calling party number to the number of forwarder. It's usually used for billing purpose. Only valid if `ForwardOnFacility=1`.

- `Q931PortRange=20000-20999`
  Default: `0` (random) Specify the range of TCP port number for Q.931 signalling channels. Note the range size may limit the number of concurrent calls.

- `H245PortRange=30000-30999`
  Default: `0` (random) Specify the range of TCP port number for H.245 control channels. Note the range size may limit the number of concurrent calls.

4.2.3 Section [Proxy]

The section defines the H.323 proxy features. It means the gatekeeper will route all the traffic between the calling and called endpoints, so there is no traffic between the two endpoints directly. Thus it is very useful if you have some endpoints using private IP behind an NAT box and some endpoints using public IP outside the box.

The gatekeeper can do proxy for logical channels of RTP/RTCP (audio and video) and T.120 (data). Logical channels opened by fast-connect procedures or H.245 tunnelling are also supported.

Note to make proxy work, the gatekeeper must have **direct connection** to both networks of the caller and callee.

- `Enable=1`
  Default: `0` Whether to enable the proxy function. You have to enable gatekeeper routed mode first (see the 4.2.2 (previous section)). You don't have to specify H.245 routed. It will automatically be used if required.
- `InternalNetwork=10.0.1.0/24`
  Default: `N/A` Define the networks behind the proxy. Multiple internal networks are allow. The proxy route channels only of the communications between one endpoint in the internal network and one external. If you don't specify it, all calls will be proxied.

  Format:

  ```
  InternalNetwork=network address/netmask[,network address/netmask,...]
  ```

  The netmask can be expressed in decimal dot notation or CIDR notation (prefix length), as shown in the example.

  Example:

  ```
  InternalNetwork=10.0.0.0/255.0.0.0,192.168.0.0/24
  ```

- `T120PortRange=40000-40999`
  Default: `0` (random) Specify the range of TCP port number for T.120 data channels. Note the range size may limit the number of concurrent calls.
- `RTPPortRange=50000-59999`
  Default: `10000-59999` Specify the range of UDP port number for RTP/RTCP channels. Note the range size may limit the number of concurrent calls.
- `ProxyForNAT=1`
  Default: `1` If yes, the gatekeeper will proxy for calls to which one of the endpoints participated is behind an NAT box. This ensure the RTP/RTCP stream can penetrate into the NAT box without modifying it. However, the endpoint behind the NAT box must use the same port to send and receive RTP/RTCP stream. If you have bad or broken endpoints that don't satisfy the precondition, you have better to disable this feature and let the NAT box forward RTP/RTCP stream for you.
- `ProxyForSameNAT=0`
  Default: `0` Whether to proxy for calls between endpoints from the same NAT box. You do not need to enable this feature in general, since usually endpoints from the same NAT box can communicate with each other.

4.2.4 Section [GkStatus::Auth]

Define a number of rules who is allowed to connect to the status port.

- `rule=allow`
  Default: `forbid` Possible values are
  - `forbid` - disallow any connection.
  - `allow` - allow any connection
  - `explicit` - reads the parameter `ip=value` where `ip` is the IP address of the peering client, `value` is `1,0` or `allow,forbid` or `yes,no`. If `ip` is not listed the parameter `default` is used.
  - `regex` - the IP of the client is matched against the given regular expression.

    Example:

    To allow client from 195.71.129.0/24 and 195.71.131.0/24:

    ```
    regex=^195\.71\.(129|131)\.[0-9]+$
    ```
  - `password` - the user has to input appropriate username and password to login. The format of username/password is the same as 4.2.16 ([Password]) section.

  Moreover, these rules can be combined by "|" or "&". For example,
  - `rule=explicit | regex`
    The IP of client must match `explicit` **or** `regex` rule.
  - `rule=regex & password`
    The IP of client must match `regex` rule, **and** the user has to login by username and password.
- `default=allow`
  Default: `forbid` Only used when `rule=explicit`.
- `Shutdown=forbid`
  Default: `allow` Whether to allow shutdown the gatekeeper via status port.

4.2.5 Section [RasSrv::GWPrefixes]

This section lists what E.164 numbers are routed to a specific gateway.

Format:

```
gw-alias=prefix[,prefix,...]
```

Note you have to specify the alias of the gateway. If a gateway registered with the alias, all numbers beginning with the prefixes are routed to this gateway.

Example:

```
test-gw=02,03
```

4.2.6 Section [RasSrv::RewriteE164]

This section defines the rewriting rules for dialedDigits (E.164 number).

Format:

```
[!]original-prefix=target-prefix[,target-prefix,...]
```

If the number is beginning with `original-prefix`, it is rewritten to `target-prefix`. Multi targets are possible. If the '`!`' flag precedes the `original-prefix`, the sense is inverted.

Example:

```
08=18888
```

If you dial `08345718`, it is rewritten to `18888345718`.

Option:

- `Fastmatch=08`
  Default: `N/A` Only rewrite dialDigits beginning with the specified prefix.

4.2.7 Section [RasSrv::PermanentEndpoints]

In this section you can put endpoints that don't have RAS support or that you don't want to be expired. The records will always keep in registration table of the gatekeeper. However, You can still unregister it via status port.

Format:

```
IP[:port]=alias[,alias,...;prefix,prefix,...]
```

Example:

For gateway,

```
10.0.1.5=Citron;009,008
```

For terminal,

```
10.0.1.10:1720=700
```

4.2.8 Section [RasSrv::Neighbors]

If the destination of an ARQ is unknown, the gatekeeper sends LRQs to its neighbors to ask if they have the destination endpoint. A neighbor is selected if its prefix match the destination or it has prefix "`*`". Currently only one prefix is supported.

Conversely, the gatekeeper only reply LRQs sent from neighbors defined in this section. If you specify an empty prefix, no LRQ will be sent to that neighbor, but the gatekeeper will accept LRQs from it.

The `password` field is used to authenticate LRQs from that neighbor. See section 4.2.15 ([Gatekeeper::Auth]) for details.

Format:

```
GKID=ip[:port;prefix;password;dynamic]
```

Example:

```
GK1=192.168.0.5;*
GK2=10.0.1.1:1719;035;gk2
GK3=gk.citron.com.tw;;gk3;1
```

4.2.9 Section [RasSrv::LRQFeatures]

Defines some features of LRQ and LCF.

- `NeighborTimeout=1`
  Default: `2` Timeout value in seconds to wait responses from neighbors. If no response from all neighbors after timeout, the gatekeeper will reply an ARJ to the endpoint sending the ARQ.

- `ForwardHopCount=2`
  Default: `N/A` If the gatekeeper receives an LRQ that the destination is either unknown, it may forward this message to its neighbors. When the gatekeeper receives an LRQ and decides that the message should be forwarded on to another gatekeeeper, it first decrements **hopCount** field of the LRQ. If **hopCount** has reached 0, the gatekeeper shall not forward the message. This options defines the number of gatekeepers through which an LRQ may propagate. Note it only affects the sender of LRQ, not the forwarder.

- `AlwaysForwardLRQ=1`
  Default: `0` Force the gatekeeper to forward an LRQ even if there is no **hopCount** in the LRQ. To avoid LRQ loops, you should use this option very carefully.

- `AcceptForwardedLRQ=1`
  Default: `1` Whether to accept an LRQ forwarded from neighbors.

- `IncludeDestinationInfoInLCF=0`
  Default: `1` The gatekeeper replies LCFs containing **destinationInfo** and **destinationType** fields, the aliases and terminal type of the destination endpoint. The neighbor gatekeeper can then save the information to suppress later LRQs. However, some vendors' gatekeepers misuse the information, thus result in interoperability problems. Only turn off this option if you encounter problems upon communicating with a third-party gatekeeper.

- `CiscoGKCompatible=1`
  Default: `0` Include a NonStandardParameter in LRQs to compatible with Cisco gatekeepers.

4.2.10 Section [RasSrv::RRQFeatures]

- `AcceptEndpointIdentifier=1`
  Default: `1` Whether to accept **endpointIdentifier** specified in a full RRQ.

- `AcceptGatewayPrefixes=1`
  Default: `1` A gateway can register its prefixes with the gatekeeper by containing **supportedPrefixes** in the **terminalType** field of RRQ. This option defines whether to accept the specified prefixes of a gateway.

- `OverwriteEPOnSameAddress=1`
  Default: `0` In some networks an endpoint's IP address may change unexpectedly. This may happen when an endpoint is using a PPP connection (e.g. modem or ADSL). This option defines how to handle a registration request (RRQ) from an IP address which does not match what we have stored. The default action is to reject the request. With this option enabled the conflicting request will cause an unregister request (URQ) to be sent for the existing IP address and the entry to be removed allowing the endpoint to register with the new address.

4.2.11 Section [RasSrv::ARQFeatures]

- `ArjReasonRouteCallToSCN=0`
  Default: `1` If yes, the gatekeeper rejects a call from a gateway to itself by reason **routeCallToSCN**.

- `ArjReasonRouteCallToGatekeeper=1`
  Default: `1` If yes, the gatekeeper rejects an answered ARQ without a pre-existing CallRec found in the CallTable by reason **routeCallToGatekeeper** in routed mode. The endpoint shall release the call immediately and re-send call Setup to the gatekeeper.

- `CallUnregisteredEndpoints=0`
  Default: `1` With this option set on, the gatekeeper will accept an ARQ from a registered endpoint with **destCallSignalAddress**, no matter the address is belongs to a registered endpoint or not. That means you can explicitly specify the IP of endpoint (registered or not) you want to call.

- `RemoveTrailingChar=#`
  Default: `N/A` Specify the trailing character to be removed in **destinationInfo**. For example, if your endpoint incorrectly contains the termination character like '#' in **destinationInfo**, you may remove it by this option.

4.2.12 Section [CallTable]

- `GenerateNBCDR=0`
  Default: `1` Generate CDRs for calls from neighbor zones. The IP and endpoint ID of the calling party is printed as empty. This is usually used for debug purpose.

- `GenerateUCCDR=0`
  Default: `0` Generate CDRs for calls that are unconnected. This is usually used for debug purpose. Note a call is considered unconnected only if the gatekeeper uses routed mode and a Q.931 Connect message is not received by the gatekeeper. In direct mode, a call is always considered connected.

- `DefaultCallTimeout=3600`
  Default: `0` Default timeout value in seconds to tear down a call. Set it to `0` to disable this feature.

4.2.13 Section [Endpoint]

The gatekeeper can work as an endpoint by registering with another gatekeeper. With this feature, you can easily build gatekeeper hierarchies. The section defines the endpoint features for the gatekeeper.

- `Gatekeeper=10.0.1.1`
  Default: `no` Define a parent gatekeeper for the endpoint(gatekeeper) to register with. Don't try to register with yourself, unless you want to be confusing. To disable this feature, set the field to be `no`.

- `Type=Gateway`
  Default: `Gateway` Define the terminal type for the endpoint. The valid values are `Gateway` or `Terminal`.

- `H323ID=CitronProxy`
  Default: `<Name>` Specify the H.323 ID aliases for the endpoint. Multiple aliases can be separated by comma.

- `E164=18888600000,18888700000`
  Default: `N/A` Define the E.164 (dialedDigits) aliases for the endpoint. Multiple aliases can be separated by comma.

- `Password=123456`
  Default: `N/A` Specify a password to be sent to the parent gatekeeper. All RAS requests will contain the password in the **cryptoTokens** field. To send RAS requests without the **cryptoTokens** field, set the field to be empty.

  Besides, the password is also used in LRQs sent to neighbor gatekeepers.
- `Prefix=188886,188887`
  Default: `N/A` Register the specified prefixes with the parent gatekeeper. Only takes effect when the Type is `Gateway`.
- `TimeToLive=900`
  Default: `N/A` Suggest a time-to-live value in second for the registration. Note that the real time-to-live timer is assigned by the parent gatekeeper in the RCF replied to the RRQ.
- `RRQRetryInterval=10`
  Default: `10` Define a retry interval in second for RRQs if no response received from the parent gatekeeper.
- `ARQTimeout=2`
  Default: `2` Define the timeout value in second for ARQs.
- `UnregisterOnReload=1`
  Default: `0` Defines whether the child gatekeeper unregisters and re-registers with it's parent when receiving a Reload command.
- `NATRetryInterval=60`
  Default: `60` Retry interval in second for NAT socket. Leave it out if you don't understand it.
- `NATKeepaliveInterval=86400`
  Default: `86400` Keepalive interval in second for NAT socket. Leave it out if you don't understand it.

4.2.14 Section [Endpoint::RewriteE164]

Once you specify prefix(es) for your gatekeeper endpoint, the parent gatekeeper will route calls with **dialedDigits** beginning with that prefixes. The child gatekeeper can rewrite the destination according to the rules specified in this section. By contrast, when an internal endpoint calls an endpoint registered to the parent gatekeeper, the source will be rewritten reversely.

Format:

```
external prefix=internal prefix
```

For example, if you have the following configuration,

```
                 [Parent GK]
                 ID=CitronGK
                 /        \
                /          \
               /            \
              /              \
            [Child GK]        [EP3]
            ID=ProxyGK        E164=18888200
            Prefix=188886
            /       \
          /          \
         /            \
      [EP1]          [EP2]
      E164=601       E164=602
```

With this rule:

```
188886=6
```

When EP1 calls EP3 by 18888200, the CallingPartyNumber in the Q.931 Setup will be rewritten to 18888601. Conversely, EP3 can reach EP1 and EP2 by calling 18888601 and 18888602, respectively. In consequence, an endpoint registered to the child GK with prefix '6' will appear as an endpoint with prefix '188886', for endpoints registered to the parent gatekeeper.

The section does not relate to the section 4.2.6 (RasSrv::RewriteE164), though the later will take effect first.

4.2.15 Section [Gatekeeper::Auth]

The section defines the authentication mechanism for the gatekeeper.

Syntax:

```
authrule=actions

 <authrule> := SimplePasswordAuth | AliasAuth | PrefixAuth | RadAuth | RadAliasAuth | ...
 <actions>  := <control>[;<ras>,<ras>,...]
 <control>  := optional | required | sufficient
 <ras>      := GRQ | RRQ | URQ | ARQ | BRQ | DRQ | LRQ | IRQ
```

A rule may results in one of the three codes: ok, fail, pass.

- `ok` - The request is authenticated by this module.
- `fail` - The authentication fails and should be rejected.
- `next` - The rule cannot determine the request.

There are also three ways to control a rule:

- `optional` - If the rule cannot determine the request, it is passed to next rule.
- `required` - The requests should be authenticated by this module, or it would be rejected. The authenticated request would then be passwd to next rule.
- `sufficient` - If the request is authenticated, it is accepted, or it would be rejected. That is, the rule determines the fate of the request. No rule should be put after a sufficient rule, since it won't take effect.

Currently supported modules:

- `SimplePasswordAuth/MySQLPasswordAuth/LDAPPasswordAuth/ExternalPasswordAuth` These modules check the **tokens** or **cryptoTokens** fields of RAS message. The tokens should contain at least generalID and password. For **cryptoTokens**, **cryptoEPPwdHash** tokens hashed by simple MD5 and **nestedcryptoToken** tokens hashed by HMAC-SHA1-96 (libssl must be installed!) are supported now. The ID and password are read from 4.2.16 ([Password]) section, MySQL database, LDAP or an external program for `SimplePasswordAuth`, `MySQLPasswordAuth`, `LDAPPasswordAuth` and `ExternalPasswordAuth` modules, respectively. Support for other backend databases is easily to add.

- `NeighborPasswordAuth` The module is used only to authenticate LRQs from neighbors defined in section 4.2.8 ([RasSrv::Neighbors]).
- `AliasAuth/MySQLAliasAuth/LDAPAliasAuth` The module can only be used to authenticate RegistrationRequest (RRQ). The IP of an endpoint with a given alias should match a specified pattern. For `AliasAuth` the pattern is defined in 4.2.19 ([RasSrv::RRQAuth]) section. For `MySQLAliasAuth`, the pattern is retrieved from MySQL database, defined in 4.2.20 ([MySQLAliasAuth]) section. For `LDAPAliasAuth` the alias (default: mail attribute) and IP (default: voIPIpAddress attribute) must be found in one LDAP entry.
- `PrefixAuth` Originally known as `GkAuthorize`. The IP or aliases of a request with a given prefix must match a specified pattern. See section 4.2.21 ([PrefixAuth]) for details. Currently the module can only authorize AdmissionRequest (ARQ) and LocationRequest (LRQ).
- `RadAuth` Provides authentication based on H.235 username/password security scheme. Authenticates RRQs and/or ARQs through remote RADIUS servers. It passes to RADIUS servers usernames and passwords extracted from CAT (Cisco Access Tokens) **tokens** carried inside RRQ and ARQ packets. Therefore if your endpoints do not support CATs or you do not need authentication scheme based on individually assigned usernames/password - this module will not work for you (but you may check `RadAliasAuth` module). See section 4.2.22 ([RadAuth]) for details.
- `RadAliasAuth` Provides authentication based on endpoint aliases and/or call signalling IP addresses with remote RADIUS servers. It does not need any H.235 **tokens** inside RAS messages, so it can be used on a wider range of systems as compared to `RadAuth`. RRQ and ARQ messages can be authenticated using this module. See section 4.2.23 ([RadAliasAuth]) for details.

You can also configure a rule to check only for some particular RAS messages. The following example configures `SimplePasswordAuth` as an optional rule to check RRQ and ARQ. If an RRQ is not checked (not contains **tokens** or **cryptoTokens** fields), it is checked by `AliasAuth`. The default is to accept all requests.

Example:

```
SimplePasswordAuth=optional;RRQ,ARQ
AliasAuth=sufficient;RRQ
default=allow
```

4.2.16 Section [Password]

The section defines the userid and password pairs used by `SimplePasswordAuth` module. Use '`make addpasswd`' to generate the utility `addpasswd`.

Usage:

```
addpasswd config userid password
```

Options:

- `KeyFilled=123`
  Default: `0` Default value to initialize the encryption key.

- `CheckID=1`
  Default: `0` Check if the aliases match the ID in the tokens.
- `PasswordTimeout=120`
  Default: `-1` The module `SimplePasswordAuth` and all its descendants will cache an authenticated password. This field define the cache timeout value in second. `0` means never cache the password, while a negative value means the cache never expires.

4.2.17 Section [MySQLAuth]

Define the MySQL database, table and fileds to retrieve the userid and password.

- `Host=localhost`
  Default: `localhost` Host name or IP of the MySQL server.
- `Database=billing`
  Default: `billing` The database to connect.
- `User=cwhuang`
- `Password=123456`
  The user name and password used to connect to the database.
- `Table=customer`
  The table in the database to query.
- `IDField=IPN`
  The field name of user id.
- `PasswordField=Password`
  The field name of password.
- `ExtraCriterion=Kind > 0`
  Default: `N/A` Specify extra criterion.

The SQL command will be issued:

```
SELECT $PasswordField FROM $Table WHERE $IDField = %id [AND $ExtraCriterion]
```

4.2.18 Section [ExternalPasswordAuth]

Specify an external program to retrieve the password. The program should accept ID from stdin and print the password to stdout.

- `PasswordProgram=/usr/local/bin/getpasswd`
  Default: `N/A` The executable of the external program.

4.2.19 Section [RasSrv::RRQAuth]

Specify the action on RRQ reception (confirm or deny) for `AliasAuth` module. The first alias (this will mostly be an H323ID) of the endpoint to register is looked up in this section. If a parameter is found the value will apply as a rule. A rule consists of conditions separated by "&". A registration is accepted when all conditions apply.

Syntax:

```
<authrules> :=  empty  |  <authrule> "&" <authrules>

  <authrule>  := <authtype> ":" <authparams>
  <authtype>  := "sigaddr" | "sigip"
  <autparams> := [!&]*
```

The notation and meaning of <`authparams`> depends on <`authtype`>:

- `sigaddr` - extended regular expression that has to match agains the "PrintOn(ostream)" representation of the signal address of the request. Example:

  ```
  sigaddr:.*ipAddress .* ip = .* c0 a8 e2 a5 .*port = 1720.*
  ```

- `sigip` - specialized form of '`sigaddr`'. Write the signalling ip adresse using (commonly used) decimal notation: "`byteA.byteB.byteC.byteD:port`". Example:

  ```
  sigip:192.168.242.165:1720
  ```

- `allow` - always accept the alias.
- `deny` - always reject the alias.

4.2.20 Section [MySQLAliasAuth]

Define the MySQL database, table and fileds to retrieve a pattern for an alias.

- `Host=localhost`
  Default: `localhost` Host name or IP of the MySQL server.
- `Database=billing`
  Default: `billing` The database to connect.
- `User=cwhuang`
- `Password=123456`
  The user name and password used to connect to the database.
- `Table=customer`
  The table in the database to query.
- `IDField=IPN`
  The field name of user id.
- `IPField=Address`
  The field name of IP address.
- `ExtraCriterion=Kind > 0`
  Default: `N/A` Specify extra criterion.

The SQL command will be issused:

```
SELECT $IPField FROM $Table WHERE $IDField = %alias [AND $ExtraCriterion]
```

4.2.21 Section [PrefixAuth]

The section defines the authentication rule for `PrefixAuth` module. Currently, only ARQs and LRQs can be authorized by this module.

First, a most specific prefix is selected according to the **destinationInfo** field of the received request. Then the request is accepted or rejected according to the matched rules with most specific netmask. If no matched prefix is found, and the `default` option is specified, the request is accepted or rejected according to that. Otherwise it is rejected or passed to next authentication module according to the module requirement.

Format:

```
prefix=authrule[|authrule|...]
```

Syntax:

```
<authrule> :=  <result> <authrule>

  <result>    := deny | allow
  <authrule>  := [!]ipv4:<iprule> | [!]alias:<aliasrule>
```

Where <`iprule`> can be specified in decimal dot notation or CIDR notation, <`aliasrule`> is expressed in regular expression. If the '`!`' flag precedes the rule, the sense is inverted.

Example:

```
555=deny ipv4:10.0.0.0/27|allow ipv4:0/0
5555=allow ipv4:192.168.1.1|deny ipv4:192.168.1.0/255.255.255.0
86=deny !ipv4:172.16.0.0/24
09=deny alias:^188884.*
ALL=allow ipv4:ALL
```

In this configuration, all endpoints except from network `10.0.0.0/27` are allow to call prefix 555 (except 5555). Endpoints from `192.168.1.0/24` are not allowed to call prefix 5555, except `192.168.1.1`. Endpoints **not** from `172.16.0.0/24` are denied to call prefix 86. Endpoints having an alias beginning with 188884 are not allowed to call prefix 09. All other situations are allowed.

4.2.22 Section [RadAuth]

This section defines configuration settings that enable RADIUS authentication based on H.235 CATs (Cisco Access Tokens) present in RRQ and ARQ RAS requests. This authentication scheme is useful only for endpoints registered at the gatekeeper.

- `Servers=SERVER1[:AUTH_PORT];SERVER2[:AUTH_PORT];...`
  Default: `N/A` RADIUS servers to be used for authentication RAS requests. The list can contain an arbitrary number of servers. The order of servers is important, because servers will be queried by the RADIUS module in the order given. If no port information is given, port number from `DefaultAuthPort` will be used. If these attributes are not set, then os services file will be queried for 'radius' services port. Servers could be IP addresses or DNS names.

  Example:

  ```
  Servers=192.168.3.1:1645;192.168.3.2:1812;radius.mycompany.com
  ```

- `LocalInterface=IP_OR_FQDN`
  Default: `N/A` Particular local network interface that RADIUS client should use in order to communicate with RADIUS servers. This parameter can be useful on NAT machines to restrict number of network interfaces used for RADIUS communication. By default this value is empty and allows RADIUS requests to be sent on any (best suitable) network interface. If you are not sure what you are doing, it is better to leave this option unset.

- `RadiusPortRange=10000-11000`
  Default: `N/A` By default (if this option is not set) RADIUS client allocates ports dynamically as specified by the operating system. If you want to restrict RADIUS client to use ports from a particular range only - set this parameter.

- `DefaultAuthPort=PORT_NO`
  Default: `An entry for 'radius' service in services file or 1812` Default port number to be used for RADIUS authentication requests (Access-Request packets), if not overriden by `Servers` attribute.

- `SharedSecret=SECRET`
  Default: `N/A (empty string)` Secret used to authenticate this GNU GK (NAS client) to RADIUS server. It should be a cryptographically strong password.

- `RequestTimeout=TIMEOUT_MS`
  Default: `2000` (miliseconds) Timeout (miliseconds) for RADIUS server response to a request sent by GNU GK. If no response is received within this time period, next RADIUS server is queried.

- `IdCacheTimeout=TIMEOUT_MS`
  Default: `9000` (miliseconds) Timeout (miliseconds) for RADIUS request 8-bit identifiers to be unique. If all 8-bit identifier range is exhausted within this period, new client socket (UDP socket) is allocation by RADIUS module. Let's take the example: we have approximatelly 60 RRQs/sec - after ca. 4 seconds 8-bit identifiers range gets exhausted - new socket allocated - after next 4 seconds the second 8-bit identifiers range gets exhauted - third socket allocated - after 9th second identifiers from the pool 1 are available again - ... . In general, too long timeout - too much resources consumed, too short timeout - RADIUS server may take incoming packets as duplicated and therefore drop it.

- `SocketDeleteTimeout=TIMEOUT_MS`
  Default: `60000` (miliseconds) - 60 s Timeout for unused RADIUS sockets to be closed. It is used in conjunction with `IdCacheTimeout` - additional sockets created during heavy GK load time periods for serving incoming requests are closed during idle periods.

- `RequestRetransmissions=NUMBER`
  Default: `2` How many times a single RADIUS request is transmissed to every configured RADIUS server (if no response is received). 1 means no retransmission, 2 - single retransmission, ... Exact retransmission method is defined by `RoundRobinServers` attribute.

- `RoundRobinServers=BOOLEAN`
  Default: `1` RADIUS requests retransmission method.

  If set to 1, RADIUS request is transmitted in the following way (until response is received):

  ```
  Server #1 Attempt #1, Server #2 Attempt #1, ..., Server #N Attempt #1
  ...
  Server #1 Attempt #RequestRetransmissions, ..., Server #1 Attempt #RequestRetransmissions
  ```

  If set to 0, the following sequence is preserved:

```
Server #1 Attempt #1, ..., Server #1 Attempt #RequestRetransmissions
...
Server #N Attempt #1, ..., Server #N Attempt #RequestRetransmissions
```

- `AppendCiscoAttributes=BOOLEAN`
  Default: `0` If set, Cisco Vendor Specific RADIUS attibutes are included in RADIUS requests (h323-conf-id,h323-call-origin,h323-call-type).

- `IncludeEndpointIP=BOOLEAN`
  Default: `1` If set, endpoint IP address is passed as Framed-IP-Address RADIUS attribute inside Access-Requests. Can be used for additional (to the username/password) authentication. For RRQs, this attribute is equal to RRQ.m_radAddress[0] or RRQ.m_callSignalAddress[0] field. For ARQs, this attribute is equal to ARQ.m_srcCallSignalAddress field or call signal address reported by the gatekeeper registration table.

4.2.23 Section [RadAliasAuth]

This section defines configuration settings that enable RADIUS authentication based on endpoint aliases and/or IP adresses present in RRQ and ARQ RAS requests. This authentication scheme is useful only for endpoints registered at the gatekeeper.

- `Servers=SERVER1[:AUTH_PORT];SERVER2[:AUTH_PORT];...`
  Default: `N/A` RADIUS servers to be used for RAS requests authentication. This list can contain an arbitrary number of servers. The order of servers is important, because servers will be queried by the RADIUS module in the order given. If no port information is given, port number from `DefaultAuthPort` will be used. If these attributes are not set, then os services file will be queried for 'radius' services port. Servers could be IP addresses or DNS names.

  Example:

  ```
  Servers=192.168.3.1:1645;192.168.3.2:1812;radius.mycompany.com
  ```

- `LocalInterface=IP_OR_FQDN`
  Default: `N/A` Particular local network interface that RADIUS client should use in order to communicate with RADIUS servers. This parameter can be useful on NAT machines to restrict number of network interfaces used for RADIUS communication. By default this value is empty and allows RADIUS requests to be sent on any (best suitable) network interface. If you are not sure what you are doing, it is better to leave this option unset.

- `RadiusPortRange=10000-11000`
  Default: `N/A` By default (if this option is not set) RADIUS client allocates ports dynamically as specified by the operating system. If you want to restrict RADIUS client to use ports from a particular range only - set this parameter.

- `DefaultAuthPort=PORT_NO`
  Default: `An entry for 'radius' service in services file or 1812` Default port number to be used for RADIUS authentication requests (Access-Request packets), if not overriden by `Servers` attribute.

- `SharedSecret=SECRET`
  Default: `N/A (empty string)` Secret used to authenticate this GNU GK (NAS client) to RADIUS server. It should be a cryptographically strong password.

- `RequestTimeout=TIMEOUT_MS`
  Default: `2000` (miliseconds) Timeout (miliseconds) for RADIUS server response to a request sent by GNU GK. If no response is received within this time period, next RADIUS server is queried.

- `IdCacheTimeout=TIMEOUT_MS`
  Default: `9000` (miliseconds) Timeout (miliseconds) for RADIUS request 8-bit identifiers to be unique. If all 8-bit identifier range is exhausted within this period, new client socket (UDP socket) is allocation by RADIUS module. Let's take the example: we have approximatelly 60 RRQs/sec - after ca. 4 seconds 8-bit identifiers range gets exhausted - new socket allocated - after next 4 seconds the second 8-bit identifiers range gets exhauted - third socket allocated - after 9th second identifiers from the pool 1 are available again - ... . In general, too long timeout - too much resources consumed, too short timeout - RADIUS server may take incoming packets as duplicated and therefore drop it.

- `SocketDeleteTimeout=TIMEOUT_MS`
  Default: `60000` (miliseconds) - 60 s Timeout for unused RADIUS sockets to be closed. It is used in conjunction with `IdCacheTimeout` - additional sockets created during heavy GK load time periods for serving incoming requests are closed during idle periods.

- `RequestRetransmissions=NUMBER`
  Default: `2` How many times a single RADIUS request is transmissed to every configured RADIUS server (if no response is received). 1 means no retransmission, 2 - single retransmission, ... Exact retransmission method is defined by `RoundRobinServers` attribute.

- `RoundRobinServers=BOOLEAN`
  Default: `1` RADIUS requests retransmission method.

  If set to 1, RADIUS request is transmitted in the following way (until response is received):

  ```
  Server #1 Attempt #1, Server #2 Attempt #1, ..., Server #N Attempt #1
  ...
  Server #1 Attempt #RequestRetransmissions, ..., Server #1 Attempt #RequestRetransmissions
  ```

  If set to 0, the following sequence is preserved:

  ```
  Server #1 Attempt #1, ..., Server #1 Attempt #RequestRetransmissions
  ...
  Server #N Attempt #1, ..., Server #N Attempt #RequestRetransmissions
  ```

- `AppendCiscoAttributes=BOOLEAN`
  Default: `0` If set, Cisco Vendor Specific RADIUS attibutes are included in RADIUS requests (h323-conf-id,h323-call-origin,h323-call-type).

- `IncludeEndpointIP=BOOLEAN`
  Default: `1` If set, endpoint IP address is passed as Framed-IP-Address RADIUS attribute inside Access-Requests. Can be used for additional (to the username/password) authentication. For RRQs, this attribute is equal to RRQ.m_radAddress[0] or RRQ.m_callSignalAddress[0] field. For ARQs, this attribute is equal to ARQ.m_srcCallSignalAddress field or call signal address reported by the gatekeeper registration table.

- `FixedUsername`
  Default: `N/A` If this parameter is set, it overwrites a value of User-Name RADIUS attribute for outgoing RADIUS request. That means every Access-Request will be authenticated as for user `FixedUsername`.

- `FixedPassword`
  Default: `N/A` If not set, User-Password is a copy of User-Name. For example, if User-Name is 'john' then User-Password will also be set to 'john'. Setting this parameter overrides this behavious and User-Password attribute will be always set to the value of `FixedPassword`.

Example 1:

```
(Neither FixedUsername nor FixedPassword set)
```

All endpoints will be authenticated using their alias as the username and the password. That means, for example, endpoint 'EP1' will be authenticated with the username 'EP1 and the password 'EP1'.

Example 2:

```
(FixedUsername not set)
FixedPassword=ppp
```

All endpoints will be authenticated using their alias and the password 'ppp'.

Example 3:

```
FixedUsername=ppp
FixedPassword=ppp
```

All endpoints will be authenticated using the username 'ppp' and the password 'ppp'.

4.2.24 Section [GkLDAP::LDAPAttributeNames]

This section defines which LDAP attribute names to use.

- `H323ID`
  The endpoint's H.323 alias. Needs to be unique within the used LDAP tree (this i why we use the mail address by default).
- `TelephonNo`
  The endpoint's E.164 alias.
- `voIPIpAddress`
  The IP address to be compared against when using `LDAPAliasAuth` For now, only a single value is allowed here.
- `H235PassWord`
  The plain text password to be compared against when using H.235 (`LDAPPasswordAuth` in Gatekeeper::Auth). For now, only a single value is allowed here.

4.2.25 Section [GkLDAP::Settings]

This section defines the LDAP server and standard LDAP client operating parameters to be used.

- `ServerName`
  Default: `ldap` The LDAP server's DNS name.
- `ServerPort`
  Default: `389` The LDAP server's TCP port (usually 389).
- `SearchBaseDN`
  Default: `o=University of Michigan, c=US` Entry point into the server's LDAP tree structure. Searches are only made below this root node.
- `BindUserDN`
  Default: `cn=Babs Jensen,o=University of Michigan, c=US` The distinguished name the gatekeeper uses to bind to the LDAP server. Leave empty if you want to access the LDAP server anonymously.

- `BindUserPW`
  Default: `ReallySecretPassword` If you specified `BindUserDN`, then specify the corresponding password to be used for binding here.
- `sizelimit`
  Default: `0` Maximum number of results the server may return in response to a single search query. The gatekeeper expects each LDAP to only yields one or zero results anyway, so this parameter is rather useless.
- `timelimit`
  Default: `0` Maximum number of seconds a query may take until it's considered as "failed".

4.2.26 Section [NATedEndpoints]

The gatekeeper can automatically detect whether an endpoint is behind NAT. However, if the detection fails, you can specify it manually in this section.

Format:

```
alias=true,yes,1,...
```

Example:

Specify an endpoint with alias 601 is behind NAT.

```
601=true
```

4.2.27 Section [CTI::Agents]

This section allows the configuration of a so called virtual queues to allow inbound call distribution by an external application. A virtual queue has an H.323 alias that can be called like an endpoint.

Upon arrival of an ARQ on a virtual queue, the gatekeeper signals a RouteRequest on the status port and waits for an external application to resond with either a RouteReject (then the ARQ will be rejected) or with RouteToAlias which leads to the ARQ being rewritten so the call will be routed to the alias (eg. acll center agent) specified by the external application.

If no answer is received after a timeout period, the call is terminated.

You can have multiple virtual queues. Just seperate the names with commas (no spaces!), eg. VirtualQueue=hotline,sales.

See the monitoring section for details on the messages and responses.

- `VirtualQueue`
  Default: none This defines the H.323 aliases for the virtual queues.
- `CTI_Timeout`
  Default: `10` Timeout in seconds for the external application to answer the RouteRequest. If no answer is received during this time an ARJ will be sent to the caller.

5 Monitoring the Gatekeeper (Reference)

5.1 Status Interface

The Status Interface is the external interface for monitoring and controlling the gatekeeper. The gatekeeper will send out messages about ongoing calls to all connected clients and it can receive commands via this interface.

The interface is a simple TCP port (default: 7000), you can connect to with telnet or another client. One example of a different client is the Java GUI, aka GkGUI.

5.1.1 Application Areas

What you do with the powers of the Status Interface is up to you, but here are a few ideas:

- Call Monitoring
- Monitoring the registered endpoints
- Graphical User Interface

  See GkGUI.
- Billing Applications

  Analyse the CDR messages and forward them to a billing application.
- Interfacing external extensions

  If you don't want to publish the source code to additinal features, just publish the core functionality and interface to it through the status interface and keep the external part private.

5.1.2 Examples

Suppose you are just interested in the CDRs (call details records) and want to process them as a batch at regular intervals.

Here is a simple Perl script (`gnugk_cdr.pl`) that starts the gatekeeper and also forks a very simple client for the Status Interface and writes just the CDRs into a logfile.

```
#!/usr/bin/perl
# sample program that demonstrates how to write the CDRs to a log file
use strict;
use IO::Socket;
use IO::Handle;

my $logfile = "/home/jan/cdr.log";
my $gk_host = "localhost";
my $gk_port = 7000;
my $gk_pid;

if ($gk_pid = fork()) {
        # parent will listen to gatekeeper status
        sleep(1);       # wait for gk to start
        my $sock = IO::Socket::INET->new(PeerAddr => $gk_host, PeerPort => $gk_port, Proto => 'tcp')
        if (!defined $sock) {
                die "Can't connect to gatekeeper at $gk_host:$gk_port";
        }
        $SIG{HUP} = sub { kill 1, $gk_pid; };   # pass HUP to gatekeeper
        $SIG{INT} = sub { close (CDRFILE); kill 2, $gk_pid; };  # close file when terminated

        open (CDRFILE, ">>$logfile");
        CDRFILE->autoflush(1);  # don't buffer output
        while (!$sock->eof()) {
                my $msg = $sock->getline();
                $msg = (split(/;/, $msg))[0];   # remove junk at end of line
                my $msgtype = (split(/\|/, $msg))[0];
                if ($msgtype eq "CDR") {
                        print CDRFILE "$msg\n";
                }
        }
        close (CDRFILE);
} else {
        # child starts gatekeeper
        exec("gnugk");
}
```

5.1.3 GUI for the Gatekeeper

There are several Graphical User Interface (GUI) frontend for the gatekeeper.

- Java GUIDeveloped by Jan Willamowius. You can monitor the registrations and calls that go through the gatekeeper. A right-click on a button gives you a popup menu for that endpoint.

  This GUI works with Java 1.0 built into most web browsers. For security reasons the GUI must be run as a standalone application or served by a web server on the same IP number as the gatekeeper (you cannot run it as an applet via a local file).

  The program is available at <http://www.gnugk.org/h323gui.html>¡C

- GkGUIA new standalone Java program developed by *Citron Network Inc.* <http://www.citron.com.tw/> It requires Java 1.4. New features include:

  – Monitor multiple gatekeepers simultaneously.

  – Two view modes: Button List and Tree List.

  – Call Detail Record(CDR) and statistics.

  – GK Status Log.

  – Different colors for different endpoint types.

  – Modify gatekeeper configuration.

  – Forcedly unregister endpoints.

  – Save and print status log and CDR.

  The GkGUI is released under GNU General Public License, available at <http://www.gnugk.org/h323develop.html##java>¡C

5.2 Commands (Reference)

The command `help` or `h` will show you a list of all available commands.

- `Reload` Reload the configuration.
- `Version`, `v` Show the version and OS information of the gatekeeper.
- `Statistics`, `s` Show the statistics information of the gatekeeper.

  Example:

  ```
  Statistics
  -- Endpoint Statistics --
  Total Endpoints: 21  Terminals: 17  Gateways: 4  NATed: 2
  Cached Endpoints: 1  Terminals: 1  Gateways: 0
  -- Call Statistics --
  Current Calls: 1 Active: 1 From Neighbor: 0 From Parent: 0
  Total Calls: 1539  Successful: 1076  From Neighbor: 60  From Parent: 5
  Startup: Fri, 21 Jun 2002 10:50:22 +0800   Running: 11 days 04:22:59
  ;
  ```

- `PrintAllRegistrations`, `r`, `?` Show all registered endpoints.

  Format:

  ```
  AllRegistrations
  RCF|IP:Port|Aliases|Terminal_Type|EndpointID
  ...
  Number of Endpoints: n
  ;
  ```

  Example:

  ```
  AllRegistrations
  RCF|10.1.1.10:1720|800:dialedDigits=Wei:h323_ID|terminal|1289_endp
  RCF|10.0.1.43:1720|613:dialedDigits=Jacky Tsai:h323_ID|terminal|1328_endp
  RCF|10.0.1.55:1720|705:dialedDigits=Sherry Liu:h323_ID|terminal|1333_endp
  Number of Endpoints: 3
  ;
  ```

- `PrintAllRegistrationsVerbose`, `rv`, `??` Show details of all registered endpoints.

  Format:

  ```
  AllRegistrations
  RCF|IP:Port|Aliases|Terminal_Type|EndpointID
  Registration_Time C(Active_Call/Connected_Call/Total_Call) <r>
  [Prefixes: ##] (gateway only)
  ...
  Number of Endpoints: n
  ;
  ```

  Example:

  ```
  AllRegistrations
  RCF|10.0.1.8:1720|Accel-GW2:h323_ID|gateway|1322_endp
  Wed, 26 Jun 2002 16:40:03 +0800 C(1/5/33) <1>
  Prefixes: 09,002
  RCF|10.1.1.10:1720|800:dialedDigits=Wei:h323_ID|terminal|1289_endp
  Wed, 26 Jun 2002 16:40:55 +0800 C(0/32/39) <1>
  ```

```
RCF|10.0.1.66:1720|716:dialedDigits=Vicky:h323_ID|terminal|1425_endp
Wed, 26 Jun 2002 16:40:58 +0800 C(1/47/53) <1>
Number of Endpoints: 2
;
```

- `PrintCurrentCalls`, `c`, `!` Show all current calls.

  Format:

```
CurrentCalls
Call No. # | CallID | Call_Duration | Left_Time
Dialed_Number
ACF|Caller_IP:Port|Caller_EPID|CRV
ACF|Callee_IP:Port|Callee_EPID|CRV
...
Number of Calls: Current_Call Active: Active_Call From Neighbor: Call_From_Neighbor \
From Parent: Call_From_Parent
;
```

  Example:

```
CurrentCalls
Call No. 29 | CallID bd c6 17 ff aa ea 18 10 85 95 44 45 53 54 77 77 | 109 | 491
Dial 0953378875:dialedDigits
ACF|10.0.1.49:1720|4048_CGK1|25263
ACF|10.1.1.1:1720|4037_CGK1|25263
Call No. 30 | CallID 70 0e dd c0 9a cf 11 5e 00 01 00 05 5d f9 28 4d | 37 | 563
Dial 0938736860:dialedDigits
ACF|10.0.1.48:1032|4041_CGK1|11896
ACF|10.1.1.1:1720|4037_CGK1|11896
Number of Calls: 2 Active: 2 From Neighbor: 0 From Parent: 0
;
```

- `PrintCurrentCallsVerbose`, `cv`, `!!` Show details of all current calls.

  Format:

```
CurrentCalls
Call No. # | CallID | Call_Duration | Left_Time
Dialed_Number
ACF|Caller_IP:Port|Caller_EPID|CRV
ACF|Callee_IP:Port|Callee_EPID|CRV
# Caller_Aliases|Callee_Aliases|Bandwidth|Connected_Time <r>
...
Number of Calls: Current_Call Active: Active_Call From NB: Call_From_Neighbor
;
```

  Example:

```
CurrentCalls
Call No. 48 | CallID 7d 5a f1 0a ad ea 18 10 89 16 00 50 fc 3f 0c f5 | 30 | 570
Dial 0225067272:dialedDigits
ACF|10.0.1.200:1720|1448_endp|19618
ACF|10.0.1.7:1720|1325_endp|19618
# Sherry:h323_ID|Accel-GW1:h323_ID|200000|Wed, 26 Jun 2002 17:29:55 +0800 <2>
Number of Calls: 1 Active: 1 From NB: 0
;
```

- `Find`, `f` Find a registered endpoint by an alias or a prefix.

  Format:

```
Find Alias
RCF|IP:Port|Aliases|Terminal_Type|EndpointID
;
```

Example:

```
f 800
RCF|10.1.1.10:1720|800:dialedDigits=Wei:h323_ID|terminal|1289_endp
;
f 801
SoftPBX: alias 801 not found!
```

- `FindVerbose`, `fv` Find details of a registered endpoint by an alias or a prefix.

  Format:

```
FindVerbose Alias
RCF|IP:Port|Aliases|Terminal_Type|EndpointID
Registration_Time C(Active_Call/Connected_Call/Total_Call) <r>
[Prefixes: ##] (gateway only)
;
```

  Example:

```
fv 02
RCF|10.0.1.100:1720|TFN:h323_ID|gateway|4037_CGK1
Wed, 26 Jun 2002 17:47:29 +0800 C(0/84/120) <1>
Prefixes: 02,09
;
```

- `UnregisterIP` Forcedly unregister an endpoint by IP and call signalling port.

  Format:

```
UnregisterIP IP[:Port]
```

  Example:

```
UnregisterIP 10.0.1.31:1720
URQ|10.0.1.31:1032|1326_endp|maintenance;
SoftPBX: Endpoint 10.0.1.31:1720 unregistered!
```

- `UnregisterAlias` Forcedly unregister an endpoint by one of its aliases.

  Format:

```
UnregisterAlias Alias
```

  Example:

```
UnregisterAlias 601
URQ|10.0.1.31:1032|1326_endp|maintenance;
SoftPBX: Endpoint 601 unregistered!
```

- `UnregisterAllEndpoints` Forcedly unregister all registered endpoints.

  Format:

  Example:

```
UnregisterAllEndpoints
URQ|10.0.1.7:1024|1325_endp|maintenance;
URQ|10.0.1.8:1024|1322_endp|maintenance;
URQ|10.0.1.32:1032|1324_endp|maintenance;
```

```
URQ|10.0.1.36:1032|1323_endp|maintenance;
URQ|10.0.1.42:1032|1318_endp|maintenance;
Done
;
```

- `DisconnectCall` Disconnect a call with given number.

  Format:

  ```
  DisconnectCall Number
  ```

  Example:

  ```
  DisconnectCall 1533
  ```

- `DisconnectIP` Disconnect all calls of an endpoint by IP and call signalling port.

  Format:

  ```
  DisconnectIP IP[:Port]
  ```

  Example:

  ```
  DisconnectIP 10.0.1.31:1720
  ```

- `DisconnectAlias` Disconnect all calls of an endpoint by one of its aliases.

  Format:

  ```
  DisconnectAlias Alias
  ```

  Example:

  ```
  DisconnectAlias 601
  ```

- `ClearCalls` Disconnect all calls on the gatekeeper.

- `GK` Show the information of the parent gatekeeper.

- `Debug` Only used for debug purpose. Options:

  - `trc [+|-|n]` Show/modify trace level.
  - `cfg SEC PAR` Read and print a config parameter in a section.
  - `set SEC PAR VAL` Write a config value parameter in a section.
  - `remove SEC PAR` Remove a config value parameter in a section.
  - `remove SEC` Remove a section.
  - `printrm VERBOSE` Print all removed endpoint records.

  Example:

  ```
  debug trc 3
  debug set RoutedMode H245Routed 1
  ```

- `Who` Show all people on the status port.

- `RouteReject` Terminate this call on a virtual queue.

  Format:

  ```
  RouteReject CallingEndpointID CallRef
  ```

  Example:

  ```
  RouteReject endp_4711 1234
  ```

- `RouteToAlias`, `rta` Route this call on a virtual queue to the specified alias.

  Format:

  ```
  RouteToAlias Alias CallingEndpointID CallRef
  ```

  Example:

  ```
  RouteToAlias Suzi endp_4711 1234
  ```

- `Exit`, `q` Quit the status port.

5.3 Messages (Reference)

The section describes the messages output to the status interface.

- `GCF|IP|Aliases|Endpoint_Type;` The gatekeeper receives a GatekeeperRequest (GRQ) and responds with a GatekeeperConfirm (GCF).
- `GRJ|IP|Aliases|Endpoint_Type|RejectReason;` The gatekeeper receives a GatekeeperRequest (GRQ) and responds with a GatekeeperReject (GRJ).
- `RCF|IP:Port|Aliases|Endpoint_Type|EndpointID;` The gatekeeper receives a RegistrationRequest (RRQ) and responds with a RegistrationConfirm (RCF).
- `RRJ|IP|Aliases|Endpoint_Type|RejectReason;` The gatekeeper receives a RegistrationRequest (RRQ) and responds with a RegistrationReject (RRJ).
- `ACF|Caller_IP:Port|Caller_EndpointID|CRV|DestinationInfo|SrcInfo|IsAnswered;` The gatekeeper receives an AdmissionRequest (ARQ) and responds with an AdmissionConfirm (ACF).
- `ARJ|Caller_IP:Port|DestinationInfo|SrcInfo|IsAnswered|RejectReason;` The gatekeeper receives an AdmissionRequest (ARQ) and responds with an AdmissionReject (ARJ).
- `DCF|IP|EndpointID|CRV|DisengageReason;` The gatekeeper receives a DisengageRequest (DRQ) and responds with a DisengageConfirm (DCF).
- `DRJ|IP|EndpointID|CRV|RejectReason;` The gatekeeper receives a DisengageRequest (DRQ) and responds with a DisengageReject (DRJ).
- `LCF|IP|EndpointID|DestinationInfo|SrcInfo;` The gatekeeper receives a LocationRequest (LRQ) and responds with a LocationConfirm (LCF).
- `LRJ|IP|DestinationInfo|SrcInfo|RejectReason;` The gatekeeper receives a LocationRequest (LRQ) and responds with a LocationReject (LRJ).
- `BCF|IP|EndpointID|Bandwidth;` The gatekeeper receives a BandwidthRequest (BRQ) and responds with a BandwidthConfirm (BCF).
- `BRJ|IP|EndpointID|Bandwidth|RejectReason;` The gatekeeper receives a BandwidthRequest (BRQ) and responds with a BandwidthReject (BRJ).
- `UCF|IP|EndpointID;` The gatekeeper receives an UnregistrationRequest (URQ) and responds with an UnregistrationConfirm (UCF).
- `URJ|IP|EndpointID|RejectReason;` The gatekeeper receives an UnregistrationRequest (URQ) and responds with an UnregistrationReject (URJ).
- `IRQ|IP:Port|EndpointID;` The gatekeeper sends an InfoRequest (IRQ) to an endpoint to query if it is still alive. The endpoint shall respond with an InfoRequestResponse (IRR) immediately.

- `URQ|IP:Port|EndpointID|Reason;` The gatekeeper sends an UnregistrationRequest (URQ) to an endpoint to cancel its registration. The endpoint shall respond with an UnregistrationConfirm (UCF).
- `CDR|CallNo|CallId|Duration|Starttime|Endtime|CallerIP|CallerEndId|` \
  `CalledIP|CalledEndId|DestinationInfo|SrcInfo|GatekeeperID;` After a call disconnected, the call detail record is shown (in one line).
- `RouteRequest|CallerIP:Port|CallerEndpointId|CallRef|VirtualQueue|CallerAlias;` Request for an external application to route an incomming call on a virtual queue.