C-BGP User's Guide
(version 1.1.20)

Bruno Quoitin
Sebastien Tandel

Computing Science and Engineering Department
Universite catholique de Louvain
Louvain-la-Neuve, Belgium

July 12, 2005

Abstract

C-BGP is an efficient BGP decision process simulator. C-BGP can be used to experiment with modified decision processes and additional BGP attributes. It can also be used to evaluate the impact of input/output policies on the routing tables of other ASes. Thanks to its efficiency, it can be used with large topologies with sizes of the same order of magnitude than the Internet.

C-BGP is open source, written in C language and has been tested on various platforms like Linux, FreeBSD and Solaris. The BGP model implemented in C-BGP is not hindered by the transmission of BGP messages on simulated TCP connections as in packet-level simulators such as SSFNet or JavaSim. The simulator supports the complete BGP decision process, import and export filters, redistribution communities and route-reflectors. It is easily configurable through a CISCO-like command-line interface. C-BGP does not model the various BGP timers (MRAI, dampening, ...) and has a simplified session management.

To make large simulations easier to perform, C-BGP is able to load interdomain topologies produced by University of Berkeley and is also able to output all the exchanged BGP messages in MRTD format so that existing analysis scripts may be re-used. The simulator can load real BGP routing tables in MRTD format and can also save the routing tables resulting from a simulation in MRTD format.

Published: 2005

Computer Science and Engineering Department
Universite catholique de Louvain
Place Sainte-Barbe, 2
1348 Louvain-la-Neuve
Belgium

The authors are particularly grateful to Steve Uhlig, Cristel Pelsser, Cedric de Launois and Olivier Bonaventure from Universite catholique de Louvain (UCL) and to Fabian Skivee, Olivier Delcourt, Simon Balon and Jean Lepropre from University of Liege (ULg) for their comments and tests. Thanks also to Wolfgang Muhlbauer and Olaf Maennel from Technical Univertity of Munich (Germany).

The following people have also been strugling with the documentation and have therefore helped to improve it in some way: Tarek Guerniche, Jean-François Paque, Virginie Van den Schrieck and Fabienne Delbrouck.

This work was supported by the European Commission within the ATRIUM project (`www.alcatel.be/atrium`) and is now being continued under the TOTEM project (`totem.info.ucl.ac.be`). This work is also supported by the e-NEXT European Network of Excellence and by a grant of France Telecom R&D.

Contents

Chapter 1

Installation

1.1. Requirements

The building process of **C-BGP** relies on the GNU **autoconf** and **automake** tools. It should therefore work on the majority of UNIX operating systems with a working C compilation chain. The building process has been tested on various UNIX systems, mainly Linux and FreeBSD workstations as well as some Solaris systems.

In order to be built, **CBGP** requires a few external libraries. The first one, **libgds** is provided with **C-BGP**. Its building and installation processes are described in Section 1.2. **C-BGP** also requires **libpcre**, the Perl Compatible Regular Expressions (PCRE) library. The **libpcre** library can be found on `http://www.pcre.org`. Information on building and installing the PCRE library can be found on the PCRE web site. You can also install the **GNU readline** library prior to building **C-BGP** if you want to use its interactive mode. The **GNU readline** library and headers are freely available from `http://www.gnu.org`.

Before proceeding with **libgds** and **C-BGP** building and installation processes, have a look at Section 1.5 where useful options of the building process are explained.

1.2. LIBGDS installation

In order to build **C-BGP**, you will need to build and install the **libgds** library. This library is freely available from the **C-BGP** web site. In order to install the **libgds** library, download its sources `libgds-x.y.z.tar.gz` (where `x.y.z` denotes the version of the library) and follow the installation procedure described below. Note that some steps of this installation will require root privileges on the host where you install the library. If you have not such privilege, please read the Section 1.4 about non-standard installation.

First, untar the archive to a temporary directory on your host:

```
[tmp]$ tar xvzf libgds-x.y.z.tar.gz
```

This should create a new directory, `libgds-x.y.z`. Move to this directory and then run **./configure**.

```
[tmp]$ cd libgds-x.y.z
[libgds-x.y.z]$ ./configure
```

The configure script should run without any problem. When it is finished, simply type **make clean** followed by **make**. That will actually build the library.

```
[libgds-x.y.z]$ make clean
[libgds-x.y.z]$ make
```

Once the build process is done, you must install the library. This step often requires that you have root privileges. To proceed with the installation, login as root and type **make install**. The default installation prefix is `/usr/local`. The installation process will install the library file under `<prefix>/lib`. It will also install the library headers under `<prefix>/include/libgds`.

```
[libgds-x.y.z]$ su
Password:
[libgds-x.y.z]# make install
```

Note: under the Linux operating system, you will need to run **/sbin/ldconfig** in order to update the linker's database of shared libraries. Running **ldconfig** requires root privilege. If you have installed the library under a non default path, you will probably need to update the configuration of the linker which is located in `/etc/ld.so.config`. If you have not the required privilege, you can use the alternative environment variable `LD_LIBRARY_PATH`. Please refer to the documentation of your operating system.

1.3. C-BGP installation

Once you have successfully setup the **libgds** library, you can start with the **C-BGP** build process. You must own the sources archive `cbgp-x.y.z.tar.gz` freely available from the **C-BGP** web site. The procedure that you must use to build and install **C-BGP** is fairly similar to the above procedure used for the **libgds** library. First, untar the archive in a temporary directory, this will create a new directory `cbgp-x.y.z`. Move to that directory and run the **./configure** script. Once the configuration script is done, run **make**.

```
[tmp]$ tar xvzf cbgp-x.y.z.tar.gz
[tmp]$ cd cbgp-x.y.z
[cbgp-x.y.z]$ ./configure
[cbgp-x.y.z]$ make clean
[cbgp-x.y.z]$ make
[cbgp-x.y.z]$ make install
```

1.4. Installation in another directory

In certain cases, you will want to install the **libgds** library under another directory than the default `/usr/local`. This will be the case if you have not the required privileges to install under the default prefix. In this case, you must use the **./configure** script with an additional parameter *–prefix=<directory>*. For instance, in order to install under your own directory `/home/user/projects`:

```
[libgds-x.y.z]$ ./configure --prefix=/home/user/projects
[libgds-x.y.z]$ make clean
[libgds-x.y.z]$ make
[libgds-x.y.z]$ make install
```

If you have installed the **libgds** library in a non-standard directory, you will most probably encounter problems during the **C-BGP** build process. The **./configure** will probably complain because it is not able to find the **libgds** library or headers. To fix this problem, you must tell the **./configure** script about the special installation path of **libgds**. This is done with the *–with-libgds-dir* parameters. The *–with-libgds-dir* tells the **./configure** what is the installation prefix of the **libgds** library (see Section 1.2). For instance, to inform the **./configure** script that you have installed the library under `/home/user/projects`, use the following command:

```
[cbgp-x.y.z]$ ./configure --with-libgds-dir=/home/user/projects \
                      --prefix=/home/user/projects
[cbgp-x.y.z] make
[cbgp-x.y.z] make install
```

1.5. Summary of options

The configuration scripts provided with the **libgds** library and **C-BGP** can take a large number of options. To get information on these options, use **./configure** with the parameter *–help*. We summarize in Table 1.1 the options that may be useful for a regular installation.

libgds building options:	
–prefix=<path>	Change the installation prefix to <path>. The default installation prefix is `/usr/local`.
C-BGP building options:	
–prefix=<path>	Change the installation prefix to <path>. The default installation prefix is `/usr/local`.
–with-pcre=<path>	Specifies the location of the **libpcre** library and headers. This is only required if they are not in the default library and header search paths.
–enable-jni *–disable-jni*	Enable/disable the JNI interface of **C-BGP**. It is disabled by default. See Chapter B for more information.
–with-jni-dir=<path>	Specifies the location of the JNI headers. This is only required if they are not in the default header search path.
–enable-readline *–disable-readline*	Enable/disable the use of the GNU **readline** library. It is enabled by default.

Table 1.1: Summary of useful **./configure** options.

Chapter 2

User interface

2.1. Command-line usage

In order to run the **C-BGP** simulator, you must type **cbgp** at the command line[1]. The **cbgp** command supports the following parameters on the command-line. The parameters are explained in Table 2.1.

cbgp [**-h**] [**-l** *<logfile>*] [**-c** *<script>* | **-i**]

The simulator can be launched in either of two modes. The first mode, selected using the *-c* parameter, is the script mode. In this mode, the simulator reads a file which contains a sequence of commands. The commands contained in the script file are used to setup a simulation. This mode is explained in Section 2.5.

The second mode, selected using the *-i*, is the interactive mode. In this mode, the simulator prompts the user for commands. This mode is intended to users who starts using the simulator. This mode is also useful for designing new simulation scripts. The interactive mode is described in Section 2.2

-h	Display the command-line options of C-BGP.
-c <script>	Run the simulator in *script mode*. Load and execute the *<script>* file. Note: without any option, the simulator works in *script mode* and commands are taken from the standard input (stdin).
-i	Run the simulator in interactive mode. Note: the simulator must be compiled with readline to support this mode.
-l <logfile>	Specified the file that must be used to record log messages. The default behaviour is to output log messages on the standard error output (stderr).

Table 2.1: Command-line options supported by **C-BGP**.

2.2. Interactive mode

When you start **C-BGP** in interactive mode, by using the *-i* parameter, the simulator prompts you with the following messages and waits for your input.

```
[user@host]$ cbgp -i
cbgp> init.
cbgp>
```

[1]The location of the **cbgp** binary must be in your PATH environment variable.

You can then type in **C-BGP** commands. All the commands are described in Chapter 3. For instance, you can enter the **print** command in order to get a message printed on the output.

```
cbgp> print ''Hello world\n''
Hello world
cbgp>
```

If you enter an unknown command, **C-BGP** will send you an error message. For instance, if you type "foo" at the prompt, the simulator will return the following error message:

```
cbgp> foo
Error: unknown command
*** command: "foo"
*** error  : "^^^"
*** expect : bgp, include, net, pause, print, quit, set, show, sim
```

The error message tells you that the entered text has not been recognized as a command. Then, three lines follow. The first one indicates the offending command. The second line indicates where the parser found an error. Finally, the third line lists the commands that you could use instead. This list of commands depends on the context you are in. For instance, suppose you typed in "bgp foo" at the prompt:

```
cbgp> bgp foo
Error: unknown command
*** command: "bgp foo"
*** error  : "bgp ^^^"
*** expect : add, assert, options, route-map, router, show, topology
```

Since "bgp" is a valid command, the parser indicates that the offending part starts at foo. The parser expects another sub command of the **bgp** command and lists these sub commands.

In interactive mode, **C-BGP** can also throw another type of error message: when you forgot to type in a parameter required by a command. In this case, the error message will indicate the name of the missing parameter. For instance:

```
cbgp> print
Error: missing parameter
*** command: "print "
*** error  : "print ^^^"
*** expect : <message>
```

2.3. Completion of commands

2.4. History of commands

Since version 1.1.18, every time you enter a command in interactive mode, it is registered in **C-BGP**'s history. Using the up/down arrows on your keyboard, you can retrieve past commands. Note that the history of commands can be stored in a file and reloaded at the next execution.

The behaviour of the **C-BGP**'s history id driven by two environment variables. These environment variables control how the history of the command-line interface is stored in a file. If the `CBGP_HISTFILE` is set, **C-BGP** will load the historic of commands from a file named ~/`.cbgp_history`. If `CBGP_HISTFILE` is not empty, the default file name is replaced by the environment variable's value.

In addition, the value of the `CBGP_HISTFILESIZE` can be set in order to limit the number of lines that will be loaded from the history file. The value of `CBGP_HISTFILESIZE` must be a positive integer value.

2.5. Script mode

In **C-BGP**, simulations are configured through scripts. A script is a sequence of **C-BGP** commands that are used to build the topology by adding nodes and links, to setup BGP sessions and to record routing information. The available **C-BGP** commands are shortly described in the following section. Before writing scripts, let's learn some particular features of the **C-BGP** scripting interface.

First, commands are grouped into functionnal classes. The **net** class contains commands related to the network and the IP layer. The commands in this class are used to build a topology of nodes and links but also to change the IP routing table of nodes, to trace the route from one node to another or even to add IP-in-IP tunnels. The **bgp** class contains commands related to the BGP protocol. The commands in this class are used to enable the BGP protocol on a particular node, to advertise local networks, to configure BGP peerings, and so on. The **sim** class contains the simulator related commands, that is commands that are used to run/stop the simulator. Finally, some commands do not belong to any of the above classes because they are general purpose commands that serve to print a message or to include a subscript.

Second, some commands are composed of a context part. That is a part of the command can be used alone to change the current command context. Let's clarify this with an example. The **bgp router X add network Y** is composed of the context part **bgp router X** which changes the command context to the commands available in router X. If the context part of the command is executed alone, the only available commands will be commands that start with the current context.

```
bgp router X
        add network Y
        add peer Z1 Z2
        ...
```

is thus equivalent to the command **bgp router X add network Y** followed by the command **bgp router X add peer Z1 Z2**.

In order to exit the current context, type the **exit** command. The parent context is restored. It is also possible to exit all the nested contexts by typing an empty command line.

Chapter 3

Commands reference

3.1. Introduction

This section describes all the available C-BGP commands. The commands are grouped into four main groups: **net**, **bgp**, **sim** and a group with miscellaneous commands.

3.2. Network related commands

---

net add node *address*

This command adds a new node to the topology. The node is identified by its IP address. This address must be unique. When created, a new node only supports IP routing as well as a simplified ICMP protocol. If you want to add support for the BGP protocol, consider using the **bgp add router** command.

---

net ntf load *fi lename*

This command loads the given NTF file into the simulator. An NTF file contains a description of a topology. Each line of the file specifies an adjacency between two nodes. The nodes are identified by their IP addresses. In addition, the file also specifies the IGP metric associated with the adjacency. It can also optionally define the propagation delay along this adjacency.

When **C-BGP** loads the NTF file, it creates all the unexisting nodes and links. It will not worry if some nodes or links already existed before the **net ntf load** command was called.

► **Input format**

```
<node-1> <node-2> <weight> [<delay>]
```

---

net node *address* ipip-enable

This command enables the support for the IP-in-IP protocol. That means that this node can behave as a tunnel end-point. If it receives encapsulated packets with the destination address of the encapsulation header being itself, it will decapsulate the packet and deliver it locally or try to forward it depending on the encapsulated header content.

net node *address* **ping** *destination*

net node *address* **record-route** *destination*

This command records the addresses of the nodes that packet sent from the source *address* traverse to reach the *destination* address.

► Output format

```
<source> <destination> <result> <list of hops>
```

where *result* is one of

SUCCESS	The destination was reachable. In this case, the list of hops is the list of the IP adresses of the traversed nodes.
UNREACH	The destination was not reachable. In this case, the list of hops is the list of IP addresses of the nodes traversed until no route was available.
TOO_LONG	The path towards the destination was too long (i.e. longer than 30 hops). This is often the symptom of a routing loop.
DOWN	there was a route to reach the destination, but a link down was found on the way. This indicates a misconfiguration or routing error. This can however occur in transient states, after a link has been brought down and the routing has not reconverged. The last node in the list of hops indicates the node adjacent to the failing link.
TUNNEL_UNREACH	The path went through a tunnel but the tunnel end-point does not support the IP-in-IP protocol. Consider using the **net node** *X* **ip-ip enable**. The last node in the list of hops is the address of the faulty node.
TUNNEL_BROKEN	The path went through a tunnel but at a point the tunnel end-point was not reachable. The last node in the list of hops is the address of the faulty node.

► Example

```
cbgp> net node 0.1.0.1 record-route 0.2.0.2
0.1.0.1 0.2.0.2 SUCCESS 0.1.0.1 0.1.0.2 0.2.0.1 0.2.0.2
```

net node *address* **route add** *prefi x next-hop metric*

This command is used to add a route towards a *prefix* into the node identified by *address*. The command specifies the route's *next-hop* and the route's *metric*.

Note. It is often more convenient to use the **net node** *X* **spf-prefix** command which computes for each node within a given prefix the shortest route according to the used metric.

net node *address* **route del** *prefi x* **{** *next-hop* | * **}**

This command removes from the node identified by *address* a route previously added with the above command. The route to be removed is identified by the destination *prefix* as well as its *next-hop*. A wildcard can be used in place of the *next-hop*. In this case, all the routes that match the *prefix* will be removed.

net node *address* show links

This command shows all the links connected to this node.

► Output format

```
<prefix> <delay> <metric> <state> <type> [<IGP option>]
```

where the *state* can be either **UP** or **DOWN**. The *type* is one of

DIRECT	The link is a direct link towards the destination, i.e. the destination is adjacent to this node.
TUNNEL	The link is a tunnel to the destination, i.e. messages that traverse this link will be encapsulated, then routed towards the tunnel end-point and hopefully decapsulated there.

and the *IGP option*, if present, can contain the **IGP_ADV** flag which means that this link is used by the "IGP", i.e. it is used in the shortest-path computation performed by the **net node** *X* **spf-prefix** command.

► Example

```
cbgp> net node 0.0.0.1 show links
0.2.0.1/32      444     444     UP      DIRECT  IGP_ADV
0.2.0.2/32      370     370     UP      DIRECT  IGP_ADV
```

net node *address* show rt { *address* | *prefi x* | * }

This command shows the content of the routing table of node *address*. The command takes one parameter to filter the output. If the filter parameter is *, all the routes are shown. If the filter parameter is an IP address, the best route that matches the given address is shown. If the filter parameter is an IP prefix, the exact route that matches the given prefix is shown.

► Output format

```
<prefix> <next-hop> <metric> <type>
```

where the route's *type* can be one of

STATIC	The route was statically installed with the **net node** *X* **route add** command.
IGP	The route was automatically computed by the **net node** *X* **spf-prefix** command.
BGP	The route was learned by BGP and selected as best.

► Example

```
cbgp> net node 0.0.0.1 show rt *
0.1.0.1/32      0.2.0.2 0       BGP
0.2.0.1/32      0.2.0.2 0       BGP
0.2.0.2/32      0.2.0.2 0       BGP
```

net node *address* **spf-prefix** *prefix*

This command computes the shortest paths from the node identified by *address* towards all the nodes which are in the given *prefix*. The metric of the computed shortest paths is equal to the sum of the IGP weights of the traversed links. The command also adds in the node's routing table an entry for each computed path. These routing entries are of type IGP.

SPT computation. The shortest paths will only be composed of links whose end-points are in the given *prefix* and which have the **IGP_ADV** flag set (see the **net node** *X* **show links** command for more information).

Note that the behaviour of this command can be slightly modified with the **igp-inter** option. See the **net options igp-inter** statement.

net node *address* **tunnel add** *end-point*

This command adds a tunnel from the given node towards the tunnel *end-point*. Messages that will be routed through this tunnel will be encapsulated, then routed to the *end-point* and decapsulated at the *end-point*. Consider using the **net node** *X* **ip-ip enable** on the destination node to enable the decapsulation of received messages.

net options igp-inter [on | off]

This command changes the destination nodes that are considered by the SPT computation performed by the **spf-prefix** statement. If the option value is **off** (default), the SPT computation only considers as destinations the nodes which strictly match the given prefix. If the option is **on**, the SPT computation also considers as destinations the nodes which are tail-ends of links that leaves the nodes which match the given prefix.

This is illustrated in Fig. 3.1. In the left part of the figure, the **igp-inter** option is **off** and the nodes which are tail-ends of the links that go outside of the considered prefix are not considered by the SPT-computation. In the right part, those destinations are taken into account during the SPT computation and routes are setup for these destinations in the nodes that belong to the prefix.

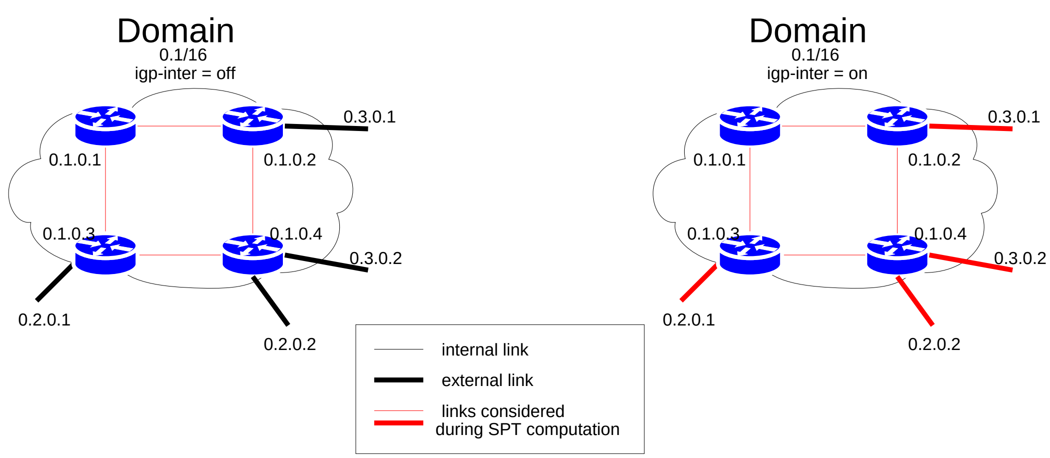

Figure 3.1: Effect of the **igp-inter** option on SPT computation.

net options max-hops *max-hops*

This command changes the maximum number of hops used by the record-route command. The default value is 15 hops.

net add link *address1 address2 delay*

This command adds a new link between two existing nodes whose addresses are *address1* and *address2* in the topology. The new link is bidirectional. The propagation delay of the link is specified by the *delay* parameter. Note also that by default, the IGP-cost of the link is fixed at the same value

net link *address1 address2* **[down | up]**

This command changes the state of a link. The link is identified using its end-points *address1* and *address2*. If the state of a link is changed, it is required to update the interdomain paths, using the **net node spf-prefix**. If the nodes run BGP, it might also be interesting to run the **bgp router rescan** command.

net link *address1 address2* **igp-weight** *weight*

This command changes the IGP weight of the link identified by the two end-points *address1* and *address2*.

Warning. This command changes the IGP weight of the link in one direction only. If you use different weights for both directions and if you use the **net node** *X* **spf-prefix** command, routing loops may be created.

3.3. BGP related commands

bgp assert peerings-ok

This command checks that all the peerings defined in all the routers are valid, i.e. for each peering, the existence of the peer is checked as well as the existence of a similar peering definition in the peer.

bgp assert reachability-ok

This command checks that all the prefixes announced by BGP are reachable by all the BGP routers.

bgp router *addr* **assert routes** *prefi x* **[best | feasible] match** *predicate*

bgp router *addr* **assert routes** *prefi x* **[best | feasible] show**

bgp add router *as-num address*

This command adds BGP support into the node identified by *address*. The node thus becomes a BGP router. The command also configures this router as a member of the domain identified by *as-num*.

bgp domain *as-num* **full-mesh**

bgp domain *as-num* **rescan**

This command rescans the BGP routes known by all the routers in the domain. This is equivalent to calling the **bgp router rescan** command for all the routers in the domain. If there is any change, the **sim run** command should be called after the rescan.

bgp domain *as-num* show routers

bgp options local-pref *pref*

This command changes the default preference given to routes that enter a domain. The default preference is 0, but can be changed to *local-pref* using this command.

bgp options med *"deterministic"* | *"always-compare"*

This command changes the behaviour of the MED-based rule of the decision process. If the argument is "deterministic", then the rule will only compare the MED of routes received from the same AS. If the argument is "always-compare", the rule will compare the MED of all routes whatever the neighbor AS is.

bgp options msg-monitor *out-fi le*

This command enables the BGP message monitoring. All the BGP messages will be written in the given *out-file*. If the file does not exist, it will be created. If it already exists, the file will be overwritten. The BGP messages will be written in MRTD format, prefixed by the IP address of the destination router.

▶ **Output format (UPDATE)**

```
dest-ip|BGP4|time|A|peer-ip|peer-as|prefix|
as-path|origin|next-hop|local-pref|med|communities
```

▶ **Output format (WITHDRAW)**

```
dest-ip|BGP4|time|W|peer-ip|peer-as|prefix
```

It should be easy to extract messages sent to a specific destination on the basis of the first field. Then, existing analysis script can be used with the MRTD output.

▶ **Example**

```
0.2.0.0|BGP4|0.00|A|0.1.0.0|1|0.1.0.0/16|1|IGP|0.1.0.0|100|0|
```

bgp options show-mode [mrt | cisco]

This command selects which output format is used to dump BGP routes. The two possible formats are **mrt** or **cisco**. With the **mrt** format, routes are dumped as with the **route_btoa** tool from the **MRTd** routing suite, using the **-m** option. In this format, the route's fields are output on a single line, using a '|' separator. With the **cisco** format, routes are shown using CISCO IOS's format.

▶ **MRTd example**

```
   3.0.0.0/8    129.250.0.85    100     10      2914 1239 80    i
```

▶ **CISCO example**

```
_|129.250.0.85|2914|3.0.0.0/8|2914 1239 80|IGP|129.250.0.85|100|10 \
    |2914:420 2914:2000 2914:3000 65504:1239||||
```

bgp router *address* add network *prefi x*

This command adds a local network that will be advertised by this router. The given network will be originated by this router.

bgp router *address* **del network** *prefix*

This command removes a local network previously added with the above command.

bgp router *address* **load rib** *filename*

This command is used to load a dump of a Routing Information Base (RIB) of a real router into the RIB of the router identified by *address*. The RIB dump must be provided in ASCII MRT format. In addition, the command performs control on the routes contained in the RIB dump.

First, the IP address and the AS number of the peer router specified in the MRT route records must correspond to the given router. Second, the IP address of the BGP nexthop must correspond to an existing peer of the router. This constraint is strong and may be relaxed in the future, since some operational configurations can not be matched.

bgp router *address* **add peer** *as-num peer-address*

This command adds a new BGP neighbor to the router identified by *address*. The peer belongs to the domain identified by *as-num* and is identified by *peer-address*. This command also configures for this neighbor default input and output filters that will accept any route. See the **bgp router** *X* **peer** *Y* **filter** commands for more information about the route filters.

bgp router *address* **peer** *peer-address* **[down | up]**

This command starts the establishment of the BGP session with the peer identified by *peer-address* (when used with **up**) or shuts down the previously established session (when used with **down**).

bgp router *address* **peer** *peer-address* **filter [in | out] add-rule**

This command adds a rule to the input (**in**) or output (**out**) filter of the peer identified by *peer-address*. See chapter 4 for more information about filters.

bgp router *address* **peer** *peer-address* **filter [in | out] insert-rule** *index*

This command inserts a rule at position *index* to the input (**in**) or output (**out**) filter of the peer identified by *peer-address*. See chapter 4 for more information about filters.

bgp router *address* **peer** *peer-address* **filter [in | out] remove-rule** *index*

This command removes the rule at position *index* from the input (**in**) or output (**out**) filter of the peer identified by *peer-address*. See chapter 4 for more information about filters.

bgp router *address* **peer** *peer-address* **filter [in | out] show**

This command shows the rules that compose the input (**in**) or output (**out**) filter of the peer identified by *peer-address*. See chapter 4 for more information about filters.

bgp router *address* **peer** *peer-address* **next-hop-self**

bgp router *address* **peer** *peer-address* **recv** *message*

This command can be used on virtual peers to feed real BGP messages. The messages must be provided in MRT format. Only UPDATE (A) and WITHDRAW (W) messages are accepted. The MRT messages syntax is as follows:

► **Syntax**

```
BGP4|<time>|A|<router>|<as>|<prefix>|<path>|<origin>|<next-hop>|<pref>|<med>|
BGP4|<time>|W|<router>|<as>|<prefix>
```

where *time* is an integer representing the time when the message was received (this field is not used by **C-BGP**). The *router* field contains the IP address of the router where the route was collected. The *as* field is the AS-number of the router where the route was collected. The *prefix* field represents the IP prefix of the route. The *path* field is the AS-path of the route. It must be a space-separated list of AS numbers. The *origin* field contains the origin of the route. The origin can be one of IGP, EGP or INCOMPLETE. The *pref* field contains the value of the local-pref attribute of the route. It must be an integer. Finally, the *med* field contains the value of the med attribute of the route. It must also be an integer.

► **Example**

```
cbgp> bgp router 0.1.0.1 peer 0.2.0.1 recv "BGP4|0|A|10.0.0.1|1|30.0.1/24|20 30|IGP|20.
```

---

bgp router *address* **peer** *peer-address* **reset**

---

bgp router *address* **peer** *peer-address* **rr-client**

This command configures the peer identified by *peer-address* as a route-reflector client. By the way, the router identified by *address* becomes a route-reflector.

---

bgp router *address* **peer** *peer-address* **soft-restart**

This command makes possible soft-restart with virtual peers. Since the routes available to virtual peers are learned through the **bgp router peer recv** command, they are lost when the session with the virtual peer is teared down. With the soft-restart option, the Adj-RIB-in corresponding to the virtual peer is not cleared when the session is teared down. These routes will still be available upon session restart.

---

bgp router *address* **peer** *peer-address* **virtual**

This command changes the peer into a virtual peer. A virtual peer is used to feed the router with real BGP messages in MRT format. The router will not maintain a BGP session with the virtual peer. Moreover, the router will not send BGP messages to the virtual peer. The virtual peer will only accept UPDATE and WITHDRAW messages with the help of the “recv” statement.

---

bgp router *address* **rescan**

This command is used to rescan the BGP routes contained in the router identified by *address*. This command must be used if the outcome of the decision process depends on the interdomain routing. The command will build the list of all prefixes known by the router. Then, for each prefix, it will decide if the decision process must be run.

---

bgp router *address* **record-route** *prefi x*

This command records the AS-level route from the given router identified by *address* towards the given *prefix*.

► **Output format**

```
<source> <destination> <status> <AS-Path>
```

► **Example**

```
cbgp> bgp router 0.0.0.1 record-route 0.1.0.1/32
0.0.0.1 0.1.0.1/32      SUCCESS 0 2 1
```

---

bgp router *address* set cluster-id *id*

---

bgp router *address* show peers

This command shows the list of peers of the router identified by *address*. For each peer, the command shows the AS-number of the peer as well as the state of the BGP session with this peer. In addition, the command also shows the options related to this peer.

The session with the peer can be in one of the following states: **IDLE**, **ACTIVE**, **ESTABLISHED** and **OPENWAIT**. If the session if in **IDLE** state, that means that it is administratively down. The administrative state of the session can be changed using the **bgp router peer up/down** commands. The session can also be in **ACTIVE** state. This state indicates that the session is administratively up but due to the current routing state of the network, it is not established. This occurs if there is no route towards the peer. The session can also be in **ESTABLISHED** state. This state indicates that the session is currently working and that BGP routes can be exchanged with the peer. Finally, the **OPENWAIT** session indicates that the session has been partially opened. An OPEN message has been sent to the peer but no answer has been received. This can be due to the OPEN message still being in the simulator's queue (if **sim run** has not been called). This can also occur in case of reachability problems between the local router and the peer.

► **Output format**

```
<peer> <as-num> <status> [<options>]
```

► **Example**

```
cbgp> bgp router 0.0.0.1 show peers
0.2.0.1 AS2     ESTABLISHED
0.2.0.2 AS2     ESTABLISHED
```

---

bgp router *address* show rib { *address* | *prefix* | * }

This command shows the routes installed into the Loc-RIB of the router identified by *address*. The command takes one parameter to filter the output. If the filter parameter is *, all the routes are shown. If the filter parameter is an IP address, the best route that matches the given address is shown. If the filter parameter is an IP prefix, the exact route that matches the given prefix is shown.

► **Output format**

```
<flags> <prefix> <peer> <pref> <metric> <AS-Path> <origin>
```

where the *flags* can contain

*	If the route's next-hop is reachable.
>	If the route is a best route and is installed in the Loc-RIB.
i	If the route is local, i.e. installed with the **add network** command.

moreover, the *origin* is one of

i	If the origin router learned this route through an **add network** statement.
e	If the origin router learned this route from an EGP protocol.
?	If the origin router learned this route through redistribution from another protocol.

Since the only way to learn a BGP route in **C-BGP** is through the **add network** statement, the *origin* will always be **i**.

▶ Example

```
cbgp> bgp router 0.0.0.1 show rib *
i> 0.0.0.1/32   0.0.0.1 0       0       null    i
*> 0.1.0.1/32   0.2.0.2 0       0       2 1     i
*> 0.2.0.1/32   0.2.0.2 0       0       2       i
*> 0.2.0.2/32   0.2.0.2 0       0       2       i
```

bgp router *address* show rib-in { *peer* | * } { *address* | *prefi x* | * }

This command shows the routes received from the selected peers of the router identified by *address*. Thus, this command shows the content of the Adj-RIB-Ins of the given router. The command takes two parameters to filter the output. The first parameter filters the peers whose Adj-RIB-Ins are shown. The parameter can be the IP address of the peer or * to show all the Adj-RIB-Ins. The second parameter can be * to show all the routes. It can also be an IP address to show only the best route that matches the given address. Finally, it can be an IP prefix. In this case, the exact route that matches the given prefix is shown.

▶ Output format

```
<flags> <prefix> <peer> <pref> <metric> <AS-Path> <origin>
```

(see the **show rib** method for more information on the fields).

▶ Example

```
cbgp> bgp router 0.0.0.1 show rib-in * *
*  0.1.0.1/32   0.2.0.1 0       0       2 1     i
*  0.2.0.1/32   0.2.0.1 0       0       2       i
*  0.2.0.2/32   0.2.0.1 0       0       2       i
*> 0.1.0.1/32   0.2.0.2 0       0       2 1     i
*> 0.2.0.1/32   0.2.0.2 0       0       2       i
*> 0.2.0.2/32   0.2.0.2 0       0       2       i
```

bgp router *address* show rib-out { *peer* | * } { *address* | *prefi x* | * }

bgp topology load *fi le*

This command loads a topology from the specified *file*. The format of the file is similar to the AS pair relationships file specified at this address. That is each line of the file specifies a relationship between two Internet domains. Based on this file, C-BGP builds a network where each domain is composed of a unique router having the IP address equal to the domain's number (AS-NUM) multiplied by 65536. For instance, the IP address of the router which composes the domain AS7018 would be 27.106.0.0. C-BGP also configures the BGP sessions between the network's routers.

▶ **Input format**

```
<domain-1> <domain-2> <relationship> [<delay>]
```

The relationship can be *0* for a peer-to-peer relationship or *1* for a provider-to-customer relationship. The optional *delay* parameter specifies the delay on the network link between the routers in the given domains.

---

bgp topology policies

This command configures the filters of the routers according to the relationships specified in the topology loaded by the **bgp topology load** command.

---

bgp topology record-route *prefi x in-fi le out-fi le*

This command records the paths towards the given *prefix* from each router specified in the *in-file* and writes those paths in the *out-file*. The *in-file* has the following format: each line contains the identifier of a domain (i.e. its AS-NUM) from which the path has to be computed or an asterisk (*) which means that the paths from all the routers have to be computed.

▶ **Output format**

```
<src-as-num> <prefix> <as-path>
```

▶ **Example** For instance, here is the result of the EBGP_2_ROUTERS example. The path from AS1 is "1" because it has advertised the prefix 0.1/16. The path from AS2 is "2 1" because it has received a BGP message with the prefix 0.1/16 from AS1.

```
1 0.1.0.0/16 1
2 0.1.0.0/16 2 1
```

---

bgp topology run

This command establishes the BGP sessions between all the routers loaded by the bgp topology load command.

3.4. Simulation related commands

---

sim options log-level <level>

This command changes the verbosity of the simulator log. The available log levels are

everything	Every log message is written.
debug	Debug and error messages are written.
warning	Error messages are written.
severe	Only severe warning and fatal error messages are written.
fatal	Only fatal error messages are written.

---

sim queue info

---

sim queue show

---

sim run

This command starts the simulator, i.e. it starts processing the queued events until no more event is available or the simulator is stopped.

3.5. General purpose commands

include <file>

This command processes the commands found in the given *file*. The processing of the given `file` will stop as soon as an error occurs. Note that, in interactive mode (see Chapter 2, Section 2.3), it is possible to use the automatic completion of the filename parameter.

pause

This command displays the message "Paused: hit 'Enter' to continue..." and consistently waits until the user press the <return> key. This command can be used in verbose simulation scripts so that the user is able to read the results.

print <message>

This command prints a *message* on the current output. The default output is stdout. The print command recognizes and interpolates the escape sequences described in Table 3.1.

`\a`	Print an *alert* to the console. Usually, this will be transformed into an audible bell.
`\e`	Prints `\033` to the console. This can be used to send ECMA-48 sequences to the console.
`\n`	Prints a new line.
`\r`	Prints a carriage return (returns to the beginning of the line).
`\t`	Prints a tabulation.

Table 3.1: Escape-characters supported by the **print** command.

set autoflush [on | off]

This command tells **C-BGP** to flush the output stream after any commands which returns information. This is important if the simulator is used by a script which waits for a response to a request. The `CBGP.pm` Perl module uses this option.

set mem-limit *amount*

This command changes the memory limit of **C-BGP** to a maximum of *amount* bytes, using the **setrlimit** system call. Normally, there is no per-process memory limitation enforced by the operating system. The memory allocation will fail when there is no more physical memory and no more virtual memory available. However, in certain situations, reaching both the physical and virtual memory limits may pose problems. Especially on some systems (on Linux for instance) where the scheduler may kill the first application that requests memory above the reached limits. Important applications may be killed even if the user who runs the simulator has no administrative privilege. We experimented such situation, so use this option!!!

show mem-limit

This command shows the current memory limitation. The command will display two memory limits: the soft and the hard limits. The hard limit will always be enforced by the operating system and can not be extended. The soft limit can be changed using the **set mem-limit** command.

show mrt *filename predicate*

This command shows the content of a BGP routing table in MRT format. The command operates directly on MRT in binary format, using Dan Ardelean's **libbgpdump** library. Parsing MRT files directly in **C-BGP** has several advantages. First, the **libbgpdump** library runs significantly faster than the **route_btoa** tool provided with the **MRTd** routing suite. Second, it is possible to use filters in order to output a subset of the BGP routes. The syntax is the same as in the BGP session filters (see Chapter. 4). Finally, it is possible to select the output format that will be used to dump the routes, using the **bgp options show-mode** command.

► **Example**

```
cbgp> show mrt rib.20050701.0009 any
   0.0.0.0/0    213.140.32.148  100     0       12956   i
   2.0.0.0/8    196.7.106.245   100     0       2905    ?
   3.0.0.0/8    207.246.129.6   100     0       11608 2914 1239 80      i
   3.0.0.0/8    129.250.0.85    100     10      2914 1239 80    i
   3.0.0.0/8    129.250.0.11    100     1       2914 1239 80    i
   3.0.0.0/8    206.186.255.223 100     0       2493 3602 1239 80       i
   (...)
cbgp> show mrt rib.20050701.0009 "path ^2914"
   3.0.0.0/8    129.250.0.85    100     10      2914 1239 80    i
   3.0.0.0/8    129.250.0.11    100     1       2914 1239 80    i
   (...)
```

show version

This command shows the version of **C-BGP**. The version information can be used to check the compatibility with an existing script. The version information displayed by the command contains a version number composed of three numerical fields (main version/sub version/release). The numerical version is followed by informal fields which inform on compilation options. For instance the version information may be followed by `[experimental]` which means that the version has been compiled using experimental features.

Chapter 4

Filters

4.1. Introduction

This chapter describes the route filtering features of **C-BGP**. Route filtering is an important part of the simulator since it is used to implement the policies of interdomain routing. **C-BGP** provides an easy to use interface to filters similar to what can be found in real routers.

In **C-BGP**, the filters can be configured differently in each router. Moreover, for each BGP router different filters can be associated to each neighbor. We also distinguish input and output filters. The first ones are used to filter the routes that are learned from neighbors while the second ones are used to filter the routes that are redistributed to neighbors.

A typical filter in **C-BGP** is a sequence of rules. Each rule being composed of two parts. The first part of one rule is a logical combination of predicates used to check if the rule applies to a route. For instance, a predicate can check is the tested route contains a given community. The second part of one rule is a set of actions that are applied to the routes matching the rule's predicates. A typical action would be to change the route's local preference.

In **C-BGP**, the filters of one router are configured using the **peer** *X* **filter [in | out]** familly of commands. The list of these commands is given in section 3.3. The **add-rule** sub-command allows to add a new rule to the given filter. Once the rule is added, the predicates and actions can be specified with the commands described in the sections hereafter. Another sub-command makes possible to insert a new rule in the sequence of rules of one filter. It is of course always possible to show the sequence of rules of one filter with the **show** sub-commands.

4.2. Predicates

Once a new rule is added or inserted, the predicates can be specified with the **match** statement. This statement takes a single parameter which is the expression of the logical combination of predicates. The syntax of this expression is described in Fig. 4.1:

Predicates	::=	Predicate
	|	Predicates '|' Predicates
	|	Predicates '&' Predicates
	|	'(' Predicates ')'
	|	'!' Predicates

Figure 4.1: Syntax of predicates.

The atomic predicates that are currently available in **C-BGP** are described below:

any

This predicate matches any route.

community is *community*

This predicate matches only the routes that contain the given *community*. The community can be written in two forms. The first form is as an integer in the range $[0, 2^{32} - 1]$. The second form is as a couple of integers in the range $[0, 2^{16} - 1]$ separated by a colon (':').

next-hop in *prefix*

This predicate matches only the routes with a next-hop contained in the given *prefix*.

next-hop is *address*

This predicate matches only the routes with a next-hop equal to *next-hop*.

path *reg-exp*

This predicate matches only the routes with an AS-path that matches the given regular expression. The syntax of *reg-exp* is similar to **grep**'s regular expressions since **C-BGP** relies on the **libpcre** library.

prefix in *prefix*

This predicate matches only the routes with a prefix which is more specific than the given *prefix*.

prefix is *prefix*

This predicate matches only the routes with a prefix which is equal to the given *prefix*.

4.3. Actions

In order to specify the second part of one rule, **C-BGP** also provides the **actions** statement. This statement takes a single parameter which is a set of atomic actions separated by a comma. The actions that are currently available in **C-BGP** are described below:

accept

This action accepts the route.

as-path prepend *amount*

This action prepends the AS-number of the router to the AS-Path of the route. The number of times prepending is performed is given by *amount*.

community add *community*

This action adds the given community to the list of communities of the route. The *community* can be specified either as a single 32-bits integer or as a couple of two 16-bits integers separated by a colon.

Special standard communities may also be specified with special identifiers: *no-export* or *no-advertise*.

community remove *community*

This action removes the given community from the list of communities of the route being filtered. The *community* can be specified as explained in the **community add** command.

community strip

This command clears the list of communities attached to the route.

deny

This commands deny the route. If this action occurs in an input filter, the route will not be considered as feasible. If this action occurs in an output filter, the route will not be redistributed to the peer concerned by the filter.

local-pref *pref*

This command changes the local-preference of the route. The new value of the LOCAL-PREF attribute is set to *pref*. Note that this command should only be used in input filters associated with peers that are in another domain (i.e. that have a different AS-number).

metric *med | "internal"*

This command changes the MED of the route. The value can be specified explicitly by passing it as an integer value to the command. Another way to set the MED value is to specify the special-value "internal". In this case, the MED value is set to the cost of the IGP path towards the route's next-hop.

red-community add prepend *amount target*

This command attaches a redistribution community to the route. This redistribution community requests that the neighbor domain perform prepending when it redistributes the route to the domain identified by *target*.

red-community add ignore *target*

This command attaches a redistribution community to the route. This redistribution community requests that the neighbor domain does not redistributes the route to the domain identified by *target*.

4.4. Example

In order to illustrate the above obscure explanations, this section presents an example of filters. The purpose of this example is to define the filters required to enforce the Internet policies, that is the customer-provider and peer-to-peer relationships.

These policies are composed of two parts. First, the filters must control the provision of transit. The usual rule is that a domain will not provide transit to its providers, a limited transit to its peers and full connectivity to its customers. This is known as the *valley-free* property.

Second, the domains usually prefer the routes learned from customers over the routes learned from providers. The last routes being also prefered over routes learned from providers. One of the reason is that such an ordering assures the convergence of BGP. Another reason is that domains get paid for the traffic that transit on the links with their customers while they must pay for the traffic that transit over links with their providers.

Such policies are easy to setup within **C-BGP**. Let's take the example topology shown in Fig. 4.2. The topology is composed of 4 domains. Domain AS1 is composed of 3 routers, R11, R12 and R13. The first router, R11, is connected to R21, the router of its provider, AS2. R12 is connected to R31, the router of its peer, AS3. Finally, R13 is connected to R41, the router of its customer, AS4.

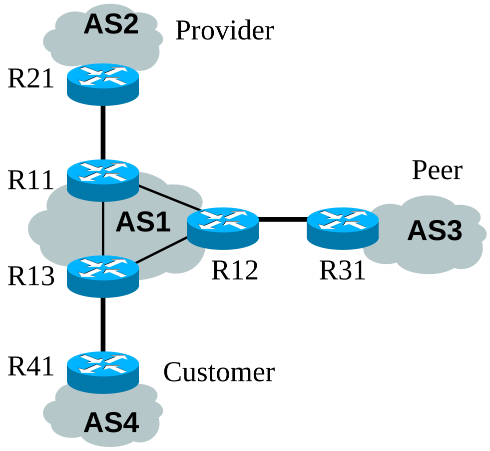

Figure 4.2: Example topology with business relationships.

The following scripts show how the various peerings are setup. Note that for convenience, the script does not contain the IP addresses of the router but their names.

```
bgp router R11
    peer 1 R12
    peer 1 R13
    peer 2 R21
    peer R21
        filter in
            add-rule
                match any
                action "local-pref 60, community add 1"
                exit
            exit
        filter out
            add-rule
                match "community is 1"
                action deny
                exit
            add-rule
                match any
                action "community strip"
                exit
            exit
        exit

bgp router R12
    peer 1 R11
    peer 1 R13
    peer 3 R31
    peer R31
        filter in
            add-rule
```

```
                match any
                action "local-pref 80, community add 1"
                exit
            exit
        filter out
            add-rule
                match "community is 1"
                action deny
                exit
            add-rule
                match any
                action "community strip"
                exit
            exit
        exit

bgp router R13
    peer 1 R11
    peer 1 R12
    peer 4 R41
    peer R41
        filter in
            add-rule
                match any
                action "local-pref 100"
                exit
            exit
        filter out
            add-rule
                match any
                action "community strip"
                exit
            exit
        exit
```

Chapter 5

Examples

5.1. Simple two-routers topology

This example describes how to build a simple topology composed of two BGP routers in two different domains. The first step is to create the nodes which correspond to the routers and the link which connects them together.

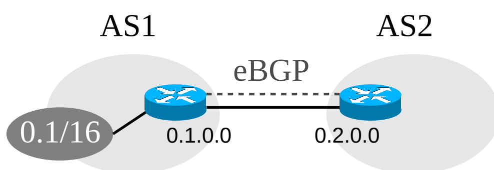

Figure 5.1: Simple two-routers topology

Building the topology is done by using the **net add** commands as explained below. The **net node X route add** statement adds static routes over the interdomain link.

```
# Setup the topology
net add node 0.1.0.0
net add node 0.2.0.0
net add link 0.1.0.0 0.2.0.0 5
net node 0.1.0.0 route add 0.2.0.0/32 0.2.0.0/32 5
net node 0.2.0.0 route add 0.1.0.0/32 0.1.0.0/32 5
```

Then, BGP has to be enabled on both nodes with the **bgp add** command. Moreover, each router has its neighbors configured and finally router 0.1.0.0 will announce a local network with BGP.

```
# Setup BGP in router 0.1.0.0
bgp add router 1 0.1.0.0
bgp router 0.1.0.0
    add network 0.1/16
```

```
    add peer 0.2.0.0
    peer 0.2.0.0 up

# Setup BGP in router 0.2.0.0
bgp add router 2 0.2.0.0
bgp router 0.2.0.0
    add peer 0.1.0.0
    peer 0.1.0.0 up
```

Finally, the simulation is started with the **sim run** command. After the simulation has converged, the BGP routing tables of both routers can be dumped.

```
# Run the simulation
sim run

# Dump both router’s routing table
print "Routing table of router 0.1.0.0\n"
bgp router 0.1.0.0 show rib *
print "Routing table of router 0.2.0.0\n"
bgp router 0.2.0.0 show rib *
```

5.2. eBGP and iBGP sessions

This example describes a somewhat more complicated configuration where 4 routers are involved. A first domain, AS1, contains a single router, 0.1.0.0 which advertises a single network 0.1/16. The second domain, AS2, contains 3 routers, 0.2.0.0, 0.2.0.1 and 0.2.0.2. There is an iBGP session between routers 0.2.0.0 and 0.2.0.2. Since there is no direct physical link between 0.2.0.0 and 0.2.0.2, we will add static routes in both routers.

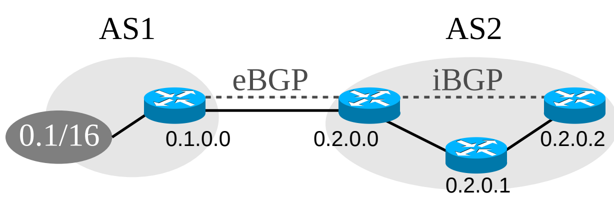

Figure 5.2: Simple topology with eBGP and iBGP sessions

So, the first step consists in building the topology:

```
# Build the topology
net add node 0.1.0.0
net add node 0.2.0.0
net add node 0.2.0.1
net add node 0.2.0.2
net add link 0.1.0.0 0.2.0.0 20
net node 0.1.0.0 route add 0.2.0.0/32 0.2.0.0 20
```

```
net node 0.2.0.0 route add 0.1.0.0/32 0.1.0.0 20
net add link 0.2.0.0 0.2.0.1 5
net node 0.2.0.0 route add 0.2.0.1/32 0.2.0.1 5
net node 0.2.0.1 route add 0.2.0.0/32 0.2.0.0 5
net add link 0.2.0.1 0.2.0.2 5
net node 0.2.0.1 route add 0.2.0.2/32 0.2.0.2 5
net node 0.2.0.2 route add 0.2.0.1/32 0.2.0.1 5
```

Then, we must add in nodes 0.2.0.0 and 0.2.0.2 a route to each other that goes through node 0.2.0.1. This is done with the **net node X route add** command.

```
# Add static routes between 0.2.0.0 and 0.2.0.2
net node 0.2.0.0 route add 0.2.0.2/32 0.2.0.1 10
net node 0.2.0.2 route add 0.2.0.0/32 0.2.0.1 10
```

Finally, the BGP protocol is enabled in routers 0.1.0.0, 0.2.0.0 and 0.2.0.2. The BGP peerings are configured and a single network is advertised by 0.1.0.0.

```
# Setup BGP in router 0.1.0.0
bgp add router 1 0.1.0.0
bgp router 0.1.0.0
    add network 0.1/16
    add peer 2 0.2.0.0
    peer 0.2.0.0 up

# Setup BGP in router 0.2.0.0
bgp add router 2 0.2.0.0
bgp router 0.2.0.0
    add peer 1 0.1.0.0
    peer 0.1.0.0 next-hop-self
    add peer 2 0.2.0.2
    peer 0.1.0.0 up
    peer 0.2.0.0 up

# Setup BGP in router 0.2.0.2
bgp add router 2 0.2.0.2
bgp router 0.2.0.2
    add peer 2 0.2.0.0
    peer 0.2.0.0 up
```

5.3. Domains and SPT computation

In the above example, we have added two static routes between node 0.2.0.0 and node 0.2.0.2. These routes were easy to add but when the topology becomes large, configuring static routes can become tedious. Fortunately, **C-BGP** provides an alternative to a manual route setup: a shortest path tree (SPT) computation. It is possible to compute the shortest-path from one node to a group of other nodes and automatically setup the required routes. Today, the only way to specify the group of destination nodes is through a network prefix, that is a prefix specifies the group of all nodes which have an IP address that matches the prefix. Usually, the prefix will cover the whole domain to which the SPT root node belongs. Indeed, an hierarchical addressing scheme must be used in order to be able to use this facility.

The command to use for the purpose of computing the shortest path tree is **net node X spf-prefix P**. The command computes the SPT rooted at node X to all the nodes in prefix P. The statements used in the above example (5.2) for the purpose of setting up static routes between each pair of nodes in the same domain, can thus be replaced by the following statements. In the case of large domains, it is a far more straightforward manner to configure the intradomain routes.

```
net node 0.2.0.0 spf-prefix 0.2/16
net node 0.2.0.1 spf-prefix 0.2/16
net node 0.2.0.2 spf-prefix 0.2/16
```

The behaviour of the **spf-prefix** statement can be slightly altered with the use of the **igp-inter** option. This option can take the values **on** and **off** (default) with the **net options igp-inter** statement. If the option value is **off**, then the SPT computation will only consider as destinations the internal nodes. If the option value is **on**, then the SPT computation will also consider as destinations the nodes which are tail-ends of interdomain links.

To clarify this, let's take the example network shown in Fig. 5.3. The network contains three domains (which can be different ASes). A typical configuration of this topology will require running an IGP inside each domain, which is modelled by the SPT computation, and BGP to provide reachability between the domains. However, in order for BGP to be run, it is required for border routers to know a route towards the tail-end of interdomain links. There are two ways to do this. The first possibility relies on the setup of static routes on the external links. This is implemented in **C-BGP** with the help of the **net node X route add** statement. In addition, eBGP sessions must be configured with the **next-hop-self** statement.

The second possibility is to insert the tail-ends of the external links in the SPT computation. It is implemented with the help of the SPT computation and the **igp-inter** option being **on**. In this case, using **next-hop-self** on eBGP sessions is useless.

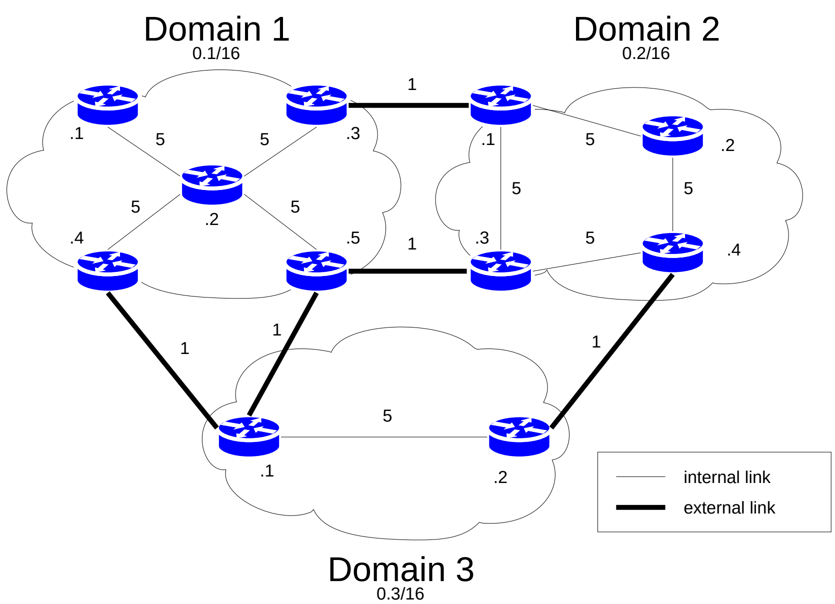

Figure 5.3: Three domains interconnected by external links.

5.4. Multi-Exit-Discriminator

In this example, we show how an AS can be configured in order to advertise routes with the MED value set based on the IGP cost to the next-hop. We also illustrate how the MED-based rule of the decision process can be changed in order to allow the comparison of the MED value of routes received from different neighbor ASes, an option found in commercial routers under the name *always-compare*.

The example topology we use is shown in Fig. 5.4. The topology contains 4 different ASes. Each AS is associated with a network with a /16 mask. For instance, AS1 is the network 0.1/16; AS2 is the network 0.2/16 and so on. There are multiple peerings between those 4 ASes. A single prefix 0.1/16 is advertised by AS1 and we observe how it is propagated until AS4, and in particular, until router 0.4.0.5 (the lowest router in the Fig. 5.4). We do not show in this example the script required to setup the different networks and BGP sessions. However, the IGP weights are shown beside the links in Fig. 5.4.

Let's now focus on the MED utilization. In our example, AS2 and AS3 advertises to AS4 routes towards AS1's prefix, 0.1/16, with the MED value set to the IGP cost towards the next-hop. In order to achieve this, we must configure output filters in the border routers of AS2 and AS3. Those output filters will contain a single rule which matches any route and whose action is to change the MED value of a matched route to the IGP cost to its next-hop. The following script shows how it is done in router 0.2.0.3 in AS2 which has a BGP session with router 0.4.0.3 in AS4. A similar configuration is made in routers 0.2.0.1, 0.3.0.1 and 0.3.0.2.

```
bgp router 0.2.0.3
    peer 0.4.0.3
        filter out
            add-rule
                match any
                action "metric internal"
                exit
            exit
        exit
    exit
```

On the other hand, we must also configure how the MED values received by routers of AS4 will be treated. The default behaviour is to only compare the MED values of two different routes if these routes have been received from the same neighboring AS. This is configured with the BGP option **bgp options med deterministic**. This statement must be issued before any decision is made by BGP, that is, before the statement **sim run** is issued to the simulator.

Another behaviour is possible for the MED-based rule with the statement **bgp options med always-compare** which means that the MED values of the routes are compared whatever the announcing AS was.

On Fig. 5.4, we show in green the route used by router 0.4.0.5 to reach destination prefix 0.1/16 when the MED-based rule only compares the MED of routes received from the same neighbor AS. When the *deterministic* mode is used (green route), the egress is 0.4.0.1. The reason for this is that routes are grouped by neighboring AS before their MED values are compared. When comparing the MED values of routes received from AS2, the one from received from 0.2.0.3 is prefered (the MED value is 1 while the MED value of the route received from 0.2.0.1 is 15). When comparing the MED values of routes received from AS3, both routes are kept because they have the same MED value, that is 5. There are thus 3 routes remaining and the decision process then relies on the router-ID. The lowest router ID is 0.4.0.1 an the route received from this router is thus prefered.

We show in brown the route used by router 0.4.0.5 when the MED-based rule compares all routes. When the *always-compare* mode is used, the MED values of the 4 routes received from both AS2 and AS3 are compared and the route received from router 0.2.0.3, which has the lowest MED value, is selected as best.

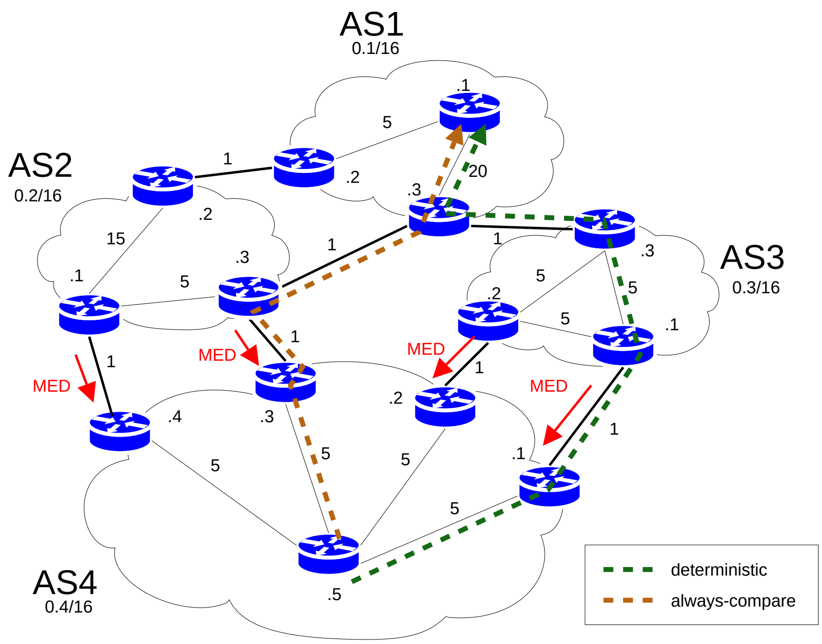

Figure 5.4: Example topology where MED is advertised.

Appendix A

Perl interface

In order to ease the development of applications or scripts that interact with **C-BGP**, a **Perl** interface is provided. This interface handles the dialog between a **Perl** script and a **C-BGP** instance. The **Perl** interface comes as a module that has to be imported in the **Perl** script. A small set of methods is defined in the module in order to establish the dialog between the **Perl** script and the **C-BGP** instance as well as to send and receive messages to and from the **C-BGP** instance.

The **Perl** interface module works as follows. First, it runs the **C-BGP** simulator and sets up new file descriptors in order to be able to write to **C-BGP**'s standard input and read from its standard output. The **Perl** module relies on a separate thread to manage the communication between the **Perl** script and the **C-BGP** instance for the purpose of avoiding buffering problems. The **Perl** script developper has thus little time to spend in order to tackle these issues. By using the methods provided by the **Perl** module, handling the interaction with **C-BGP** only takes a few lines of code.

A.1. Installation

In order to install the **CBGP.pm** module, it must be copied in a directory that **Perl** knows. To make the installation process easier, a standard installation procedure is provided with the **CBGP.pm** package. Use the following steps to proceed with the installation:

```
[perl_CBGP-x.x]$ perl Makefile.PL
Writing Makefile for CBGP
[perl_CBGP-x.x]$ make install
```

If you want to install the **CBGP.pm** module in a non-standard or a private directory, then you must define the `PERLLIB` environment variable so that it contains the path where you have installed the module.

A.2. Initialization

The following script illustrates how a basic **Perl**-**C-BGP** interaction can be setup. The first line imports the CBGP module with the version being at least 0.3 (the version we are talking about in this section). Note that in order to use this version of the CBGP module, we recommend that you use a version of **Perl** that is no older than 5.8.3 since previous versions suffer from various bugs related to multi-threading.

```
use CBGP 0.3;

my $cbgp= new CBGP;
$cbgp->spawn();
$cbgp->send("set autoflush on\n");
...
```

```
# Interaction with the C-BGP instance
...
$cbgp->finalize();
```

The second line creates an instance of the CBGP object which will be responsible for handling the interaction with the **C-BGP** instance. Then, the **spawn** method launches the **C-BGP** simulator and establishes the dialog. The third line configures **C-BGP** so that it will flush most of it output messages. This is required in order to function in an interactive manner. At this point, the script can use the **send** and **expect** methods to send/receive commands to/from the **C-BGP** instance. Finally, the last line finalize the interaction. It shuts down the thread and closes the input descriptor of the **C-BGP** instance which, as a consequence, terminates the simulator.

A.3. Interaction

Once the CBGP module has been initialized and a CBGP object has been created, the developper can send and receive messages to/from the **C-BGP** simulator. The commands that can be sent to **C-BGP** are the same commands as in **C-BGP** scripts (see Ch. 3). For instance, the following script creates a tiny topology composed of two nodes and one link. It then traces the route from one node to the other.

```
$cbgp->send("net add node 0.1.0.1\n");
$cbgp->send("net add node 0.1.0.2\n");
$cbgp->send("net add link 0.1.0.1 0.1.0.2 5\n");
$cbgp->send("net node 0.1.0.1 spf-prefix 0.1/16\n");
$cbgp->send("net node 0.1.0.2 spf-prefix 0.1/16\n");
$cbgp->send("net node 0.1.0.1 record-route 0.1.0.2\n");
my $answer= $cbgp->expect(1);
my @fields= split /\s+/, $answer, 4;
if ($fields[2] eq 'SUCCESS') {
  print "Route: ".$fields[3]."\n";
} else {
  print STDERR "Error: could not trace the route\n";
}
```

The first five lines create two nodes, one links and initialize the routing tables of both nodes. The sixth line requests the simulator to trace the route from the first node to the other. The **Perl** script then waits for the answer with the help of the **expect** method. The parameter '1' given to the **expect** method means that the call is blocking, i.e. this call will block until something has been received from **C-BGP**. Once the answer has been received, the **Perl** script prints the route to the standard output. Note that the format of the answer to a **record-route** statement is described in Ch. 3.

A.4. Checkpoints

It will often be required during a dialog with the **C-BGP** simulator to receive the answer from a statement that produces an answer composed of an unknown number of lines. This is the case with commands such as **show rt** or **show rib**. The problem with these commands is that you cannot use a blocking call to the **expect** method since you do not know when the simulator has sent the whole answer.

Fortunately, a simple solution exists that makes possible to circumvent this issue, the use of checkpoints. Checkpoints are places in the dialog where a synchronization is required. For instance, when you need to know that the simulator has completed a request or when you need that the simulator signals that it has finished producing its multiple lines answer, you will use checkpoints.

The simplest way to implement checkpoints with the CBGP module is to request **C-BGP** to write to its output a message that you will be able to catch. For instance, after you have requested **C-BGP** to dump the routing table of a node, you ask it to write a message "DONE" to its output. This example is illustrated in the following excerpt of a **Perl** script.

```
$cbgp->send("net node 0.1.0.1 show rt *\n");
$sbgp->send("print \"DONE\\n\"\n");
while ((my $result= $cbgp->except(1)) ne "DONE") {
  ...
  # Process $result...
  ...
}
```

A.5. Logging

Since it will sometimes be difficult to debug the interaction between your **Perl** scripts and **C-BGP**, the module provides a simple way to log all the commands that were sent to the **C-BGP** instance. In this way, you are able to load the log file afterwards into **C-BGP** using the interactive mode in order to debug your scripts.

By default, this option is turned off. To turn it on, use the following **Perl** command: **$cbgp->log= 1**. The consequence is that every subsequent command that is sent to the **C-BGP** instance will be written in the file .CBGP.pm.log in the working directory. Note that this file is removed each time the command **$cbgp->new()** is used.

Appendix B

Java Native Interface (JNI)

B.1. Introduction

In order to ease the development of **Java** applications interacting with **C-BGP**, a *Java Native Interface* (JNI) is provided with **C-BGP**. This interface allows a direct interaction with **C-BGP**. The JNI comes as a jar archive that has to be imported in the **Java** application.

This interface is still under development. Therefore, all the commands available with **C-BGP** scripting (see Chapter 3) are not yet available through the JNI.

B.2. Installation

C-BGP can be compiled with a Java Native Interface (JNI) in order to be linked with Java applications. For this purpose, **C-BGP** comes with a Java package. In order to compile **C-BGP**'s JNI package, you need a Java Software Development Kit (SDK) correctly installed on your computer. You also need to give the *–enable-jni* option to the **./configure** script.

Once you have compiled and installed **C-BGP**, a **jar** package `CBGP.jar` and a dynamic library `libcsim.so` will be installed in `<prefix>/lib` where `<prefix>` is the installation directory of **C-BGP**. You can then update your `CLASSPATH` environment variable with the full path of the **jar** package. With a **bash** shell, this can be done using the **export** command. For instance, the following line could be added to the `.bash_profile` file into your home directory:

```
[user@host]$ export CLASSPATH=$CLASSPATH:<prefix>/lib/CBGP.jar
```

You may also need to tell the linker that the **libcsim.so** library is available by either updating your `/etc/ld.so.conf` file under Linux or by adding the path to the library to your `LD_LIBRARY_PATH` environment variable or by giving the *-Djava.library.path=<prefix>/lib* parameter to the Java Virtual Machine (JVM).

B.3. Description of the API

The Java classes provided within the `CBGP.jar` archive are part of the `be.ac.ucl.ingi.cbgp` package. The main class is called **CBGP** and it currently contains all the native methods. Most of the methods described in this section follow the same naming rules and have the same semantic than the ones described in Chapter 3. All the methods have the **public** and **native** attributes. These attributes are not written at the head of each command in order to enhance the readability of the documentation.

In order to use the simulator's library, your Java application must call the **init** method before using any of the methods of the **CBGP** class. When your application terminates, it must call the **destroy** method.

void init(String *SLogFile*)

This method must be used to initialize the **C-BGP** library. The method takes a unique argument which specifies where the **C-BGP** library will write its log messages. An example file could be `/tmp/cbgp_jni.log`.

void destroy()

This method cleans everything in the **C-BGP** library. It is supposed to free all the memory allocated during the simulation. This method should be used once the Java application does not need the **C-BGP** library anymore.

B.4. Network related methods

int netAddNode(String *sAddr*)

This method adds a new node to the topology. The node is identified by its IP address. This address must be unique. When created, a new node only supports IP routing as well as a simplified ICMP protocol. If you want to add support for the BGP protocol, consider using the **bgpAddRouter** method.

int netAddLink(String *sSrcAddr*, String *sDstAddr*, int *iDelay*)

This method adds a new link between two existing nodes whose addresses are *sSrcAddr* and *sDstAddr* in the topology. The new link is bidirectional. The propagation delay of the link is specified by the *iDelay* parameter. Note also that by default, the IGP-cost of the link is fixed at the same value

int netLinkWeight(String *sSrcAddr*, String *sDstAddr*, int iWeight)

This method changes the IGP weight of the link identified by the two end-points *sSrcAddr* and *sDstAddr*. To the contrary of the script command **net link igp-weight** (see 3.2), this method changes the IGP weight of the link for both directions).

int netLinkUp(String *sSrcAddr*, String *sDstAddr*, boolean *bUp*)

These method gives the possibility to change the availability of a link. By using the first method we enable the use of a link identified by its two end-points. By using the second method it's possible to disable a link. Note that, as the IGP is not dynamic, when a change is made at the link level, the application has to recompute the shortest paths (see the method **netNodeSpfPrefix ??**).

int netNodeRouteAdd(String *sNodeAddr*, String *sPrefi x*, String *sNexthop*, int *iMetric*)

This method is used to add a route towards a *sPrefix* into the node identified by *sNodeAddr*. The method specifies the route's *sNextHop* and the route's *iMetric*.

Note. It is often more convenient to use the **nodeSpfPrefix** method which computes for each node within a given prefix the shortest route according to the used metric.

int netNodeSpfPrefi x(String *sAddr*, String *sPrefi x*)

This method computes the shortest paths from the node identified by *net_addr* towards all the nodes which are in the given *prefix*. The metric of the computed shortest paths is equal to the sum of the IGP weights of the traversed links. The method also adds in the node's routing table an entry for each computed path. These routing entries are of type IGP.

SPT computation. The shortest paths will only be composed of links whose end-points are in the given *prefix* and which have the **IGP_ADV** flag set (see the **nodeShowLinks** method for more information).

Vector netNodeGetLinks(String *sAddr*)

This method returns a vector of **Link** objects. The **Link** class is defined in the **CBGP.jar** archive. Each **Link** object contains the attributes of one link of the node identified by *sAddr*. See the documentation of the **Link** class for mode information.

Vector netNodeGetRT(String *sNodeAdd*, String *sPrefi x*)

This method returns the content of the routing table of node *sNodeAddr*. This method is similar to the **net node show rt** described in Chapter 3. The method returns a Vector of IPRoute objects.

B.5. BGP related methods

int bgpAddRouter(String *sName*, String *sRouterAddr*, int *iASNumber*)

This method adds BGP support into the node identified by *sRouterAddr*. The node thus becomes a BGP router. The method also configures this router as a member of the domain identified by *iASNumber* and adds a *sName* to it.

int bgpDomainRescan(int *iASNumbe*)

This method is similar to the **bgp domain rescan** command described in Chapter 3.

int bgpRouterNetworkAdd(String *sRouterAddr*, String *sNetwork*)

This method adds a local network that will be advertised by this router. The given network will be originated by this router.

int bgpRouterNeighborAdd(String *sRouterAddr*, String *sPeerAddr*, int *iASNumber*)

This method adds a new BGP neighbor to the router identified by *sRouterAddr*. The peer belongs to the domain identified by *iASNumber* and is identified by *sPeerAddr*. This method also configures for this neighbor default input and output filters that will accept any route.

void bgpRouterNeighborNextHopSelf(String *sRouterAddr*, String *sPeerAddr*)

This method is used to change the router's behavior when updating the next-hop attribute. If the route announced to the other peer is the router received by *sRouterAddr* then the next-hop of this route will be replaced by the address of the router (*sPeerAddr*).

int bgpRouterPeerUp(String *sRouterAddr*, String *sPeerAddr*, boolean *bUp*)

This method changes the status of the BGP session with the peer identified by *sPeerAddr*. If *bUp* is true, the session is established. Otherwise, the session is teared down. This command is similar to the following commands: **bgp router peer up** and **bgp router peer down**.

int bgpRouterRescan(String *sRouterAddr*)

This method rescans RIBs of the domain's routers. It is similar to the **bgp domain rescan** described in Chapter 3.

Vector bgpRouterGetRib(String *sRouterAddr*, String *sPrefix*)

This method returns the routes installed into the Loc-RIB of the router identified by *sRouterAddr*. The command is similar to the **bgp router show rib** command described in Chapter 3. The BGP routes are returned in a Vector of BGPRoute objects.

Vector bgpRouterGetAdjRib(String *sRouterAddr*, String *sNeighborAddr*, String *sPrefix*, boolean *bIn*)

This method returns the routes received from (advertised to) the selected peers of the router identified by *sRouterAddr*, i.e. the content of the Adj-RIB-Ins (Adj-RIB-Outs) of the given router. If *bIn* is true, the content of the Adj-RIB-Ins is returned. Otherwise, the content of the Adj-RIB-Outs is returned.

The command is similar to the **bgp router show rib-in** command described in Chapter 3. The BGP routes are returned in a Vector of BGPRoute objects.

B.6. Simulation related methods

int simRun()

This command starts the simulator, i.e. its starts processing the queued events until no more event is available or the simulator is stopped.

B.7. General purpose methods

void runCmd(String *sCommand*)

This method can be used to directly send commands to the **C-BGP** simulator. The acceptable commands are listed in Chapter 3.

B.8. IPAddress class

B.9. IPPrefix class

B.10. Link class

B.11. Route interface

B.12. IPRoute class

B.13. BGPRoute class

Appendix C

GNU Lesser General Public License

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages–typically libraries–of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make

exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS