# BGPmon:
# Administrator's Reference Manual

Colorado State University

February 3, 2011

# Contents

## List of Figures

## List of Tables

# 1 Introduction

BGP Monitor (BGPmon) is a light-weight, scalable, and extensible system for monitoring BGP routing. BGPmon collects routing data by imitating a real BGP router and peering with other BGP routers. All routing updates received from the peers are converted into a convenient XML format and output to interested clients. In addition, BGPmon can output periodic routing tables and control messages. BGPmon can also label routing data to simplify later analysis.

BGPmon clients receive the data from one or more BGPmon instances and perform a variety of data parsing tasks. Any program that can establish a TCP connection and parse XML can become a client. Some clients archive updates to disk, other clients identify and forward only updates that meet some criteria, and so forth. A number of BGPmon clients are provided as part of the base distribution.

## 1.1 BGPmon Design Overview

The BGPmon design extends the scalable event driven architecture in [3] to meet the requirements of BGP monitoring. BGPmon is a stream based monitoring systems that receives BGP messages from peer routers, performs some optional labeling, converts the data to an XML format, and passes the resulting data to clients via TCP connections. BGPmon clients receive an event stream in real-time or may read historical event streams from archival sources. The single stream incorporates both incremental BGP update messages and periodic routing table snapshots. BGPmon uses XML to provide easy extensibility, integrate with common tools, and allow local data annotations.

This document is intended for readers interested in installing, configuring, and using BGPmon.

Readers interested in the underlying design philosophy of the BGPmon system should refer to the technical paper[1]. Readers interested in understanding the implementation details of BGPmon or modifying the source code should refer to the detailed technical specification found in [2].

## 1.2 BGPmon Labeling

In the BGPmon design, nearly all data processing is assumed to occur at the clients. However, BGPmon can be configured to add labels to each update received from a BGP peer router. The labels produced by BGPmon include:

| | |
|---|---|
| *NEWANN* | a NEW ANNouncement (prefix not previously reachable) |
| *DPATH* | an update announcing a Different AS PATH |
| *SPATH* | an update announcing the Same AS PATH, but some other change |
| *DUPANN* | a DUPlicate ANNouncement (no change in any attribute) |
| *WITH* | a WITHdraw (prefix no longer reachable) |
| *DUPWITH* | a DUPlicate WITHdraw (unreachable prefix withdrawn) |

In order to calculate labels, BGPmon stores RIB_IN tables from each peer. But storing RIB_IN tables can consume substantial memory and add some minor additional computational costs. RIB_IN tables can be disabled on a per peer basis, but labeling is not possible if RIB_IN tables are disabled. Labeling is discussed further in Section 4.

## 1.3 BGP Monitor Chains

A single BGPmon instance can typically handle a large number of peers and clients. However, it may be useful to run multiple instances of BGPmon and connect them together into a single chain. Data from one BGPmon can be fed into an second BGPmon instance. Clients are oblivious to chains and, to client, it appears as though all data is being collected by a single BGPmon instance. BGPmon chains provides a range powerful options for scaling to vast numbers of peers, adding additional robustness, or separating peering and collector functions.

For example, suppose an administrator in Denver wanted to monitor BGP routers at exchange points in Los Angeles and London. A single BGPmon instance running in Denver could peer with routers at both the Los Angeles and London exchange points. However, this requires the use of multi-hop BGP and if the BGPmon instance fails, all data is lost. An alternate strategy would be to deploy a BGPmon instance in London and second BGPmon instance in Los Angeles. The London and Los Angeles BGPmon instances could chain to a third BGPmon instance in Denver. The LA and London BGPmon instances feed data to the Denver instance which in turns provides data to clients. Clients are unaware if the data received comes from a single instance or chain of BGPmons. Section 5 describes how to configure BGPmon chains.

## 1.4 BGP Monitor Configuration

BGPmon is designed to imitate a router and its configuration is similar to that of major router vendors. An administrator logs into BGPmon and, after passing authentication checks, has access to the administrative portion of BGPmon. Similar to the BGP routers sold by large vendors, there are two access levels. Initially, an administrator is connected in *access mode*. Access mode allows the administrator to view statistics, show routing tables, and generally view (but not change) configuration parameters. BGPmon may be configured to allow arbitrary users to login and display statistics, similar to what a user could do if granted access to a router at an ISP.

In order to change the configuration, the user must switch to *enable mode*. An administrator that has entered *enable mode* can perform all BGPmon configuration actions such as adding, deleting, or modifying BGP peers, disabling clients, setting access control policies, and so forth. All changes are stored in memory and will be lost if BGPmon restarts. At any time, an administrator in *enable mode* can save the current BGPmon configuration to a file. The configuration file can then be loaded if BGPmon restarts.

If no configuration file is specified at boot time, BGPmon attempts to load a default configuration filename. If no configuration is specified and no default configuration file is found, BGPmon allows the administrator to login on port 50,000. An alternate default port number can also be specified as a command line parameter to bgpmon.

## 1.5 BGP Monitor Clients

The basic premise is that the BGPmon server simply collects and reports data. BGPmon does not archive data or perform more complex analysis of the data. BGP Monitor clients perform a wide variety of tasks. A client simply receives XML data from BGPmon via TCP connection and then performs the desired data processing tasks. Users can build their own clients to suit individual needs. A small number of sample clients and common used clients are included in the BGPmon distribution. Clients provided in the base distribution include:

- *BGP Data Archive Client:* logs the updates received from BGPmon to disk and also periodically writes routing tables (RIB_INs) to disk.
- *BGP Data Statistics Client:* creates and maintains HTML pages that track the status of BGPmon. The results can be displayed on a website to provide on-line tracing of BGPmon behavior.
- *BGP Data Real-Time Filter:* receives updates from BGPmon and passes only the updates matching a configured criteria to downstream clients. This allows users to select a set of interesting updates from the potentially vast volume of messages sent by BGPmon.
- *BGP Sample Client:* receives updates from BGPmon and acts as starting point for designing and implementing additional site-specific clients.

## 1.6 Organization of This Document

Section 2 describes how to install the base BGPmon software and describes the resource requirements for BGPmon. Section 3 describes how to login and modify the BGPmon configuration. Section 4 describes the

BGPmon labeling options. Section 5 describes how to configure multiple BGPmon instances into a chain. Section 6 describes BGP Monitor clients, including both standard clients included in the base package and guidelines for building new clients. Section 7 provides troubleshooting help and Section 8 acknowledges the many people and organizations that helped develop BGP Monitor.

Appendix A describes the full set of commands available to BGPmon administrators.

# 2 Installing BGPmon

## 2.1 Hardware and CPU Requirements

The BGPmon server does not require specialized resources and is a designed to run on an off the shelf unix box. Note that unlike a typically router implementation, BGPmon does not implement any routing polices, perform any route selection, manage forwarding tables, or forward packets. The BGPmon server simply maintains BGP peering sessions, optionally stores routing tables (RIB_INs) for some peers, and passes XML data to clients via TCP. The processing requirements are minimal and typical boxes can support large numbers of peer connections.

BGP Monitor clients connect to BGPmon server via TCP and thus can run on the same machine or on remote machines. For example, anyone can simply telnet to the appropriate BGPmon port and, assuming they meet the access requirements configured by the administrator, begin receiving BGP data in XML format. More sophisticated clients perform more resource intensive data analysis and archiving tasks. Any program that can establish as TCP connection can become a client. Again no specialized resources are required and clients in the base distribution run on an off the shelf unix box.

In a recommended configuration, a single box runs the BGPmon server along with a data archiving client to store BGP data for later analysis. Any number of additional clients can connect to BGPmon and receive BGP data in real-time. Adding additional BGP peers and/or clients increases the resources requirements. Tests at Colorado State University have included dozens of peers and hundreds of clients running on a single "off the shelf" server running the Ubuntu operating system.

## 2.2 Memory Requirements

In its simplest configuration, BGPmon has very little mandatory state and thus can operate with a very small memory footprint. A small amount of state is kept for each peering session. As messages arrive from a peer, they are temporarily stored in internal buffers as the messages move from the peering router to the TCP output to clients. Even with hundreds of peers, BGPmon requires only a few megabytes of memory.

BGPmon memory requirements can increase dramatically when routing table (RIB_IN) storage is enabled. As updates are received from a peer, BGPmon can optionally keep track of the peer's current routes in a RIB_IN table. The RIB_IN tables are the primary memory requirements associated with BGPmon. As a general rule of thumb, a peer announcing 250,000 routes will require a 25 MB RIB_IN table

Disabling RIB_IN storage for a peer decreases memory requires, but also prevents BGPmon from labeling updates received from that peer. By comparing an update to the current RIB_IN table, the BGPmon server determines whether the update is a new announcement, path change, duplicate update and so forth as discussed above. This labeling is only possible if RIB_IN storage is enabled for the peer.

The RIB_IN may be used to periodically report the table to clients if the peer does not support BGP route refresh. In most cases, BGPmon periodically re-announces the peers routing table to clients by requesting a route refresh from the peer. If the peer does not support route refresh, BGPmon attempts to imitate a route refresh by reporting the RIB_IN file. Periodic routing tables will not be available if the peer router fails to support route refresh and BGPmon has been configured to not store RIB_IN tables for the peer.

## 2.3 Operating System Requirements

BGPmon and its clients were developed on Ubuntu and Fedora systems with the objective of being platform independent.

Ports to other unix operating systems are encouraged and some limited help is available from the BGP Monitor team.

In the BGPmon source code, the file *README_PORTS* lists other successful ports and provides instructions for porting to other operating systems.

## 2.4 Installation

To install the BGP Monitor package, use:

```
configure
make
make install
```

This builds the BGPmon server and installs the server bgpmon. In addition, the BGPmon clients discussed in Section 6 are also built and installed.

On a default unix install, the BGPmon server is installed in */usr/local/sbin/bgpmon* and clients are installed in */usr/local/bin/bgpmon_clientname*.

## 2.5 Launching BGPmon

To start BGPmon, simply launch the server using:

```
bgpmon
```

By default, bgpmon tries to load a configuration file called *bgpmon_config.txt*. If no pre-existing configuration file is found, the server starts by listening administrator login on port 50,000.

*If this is the first time you are using BGPmon*, there is no existing configuration file and server is waiting for an administrator to login on port 50,000. The login port can also be set using command line arguments described in the next subsection. Using the steps discussed in Section 3, an administrator can login and add BGP peers, set access control rules for both future configuration and client access, create BGPmon chains, and apply a variety of optional settings. At a minimum, an administrator will need to configure BGPmon to receive data from at least on BGP peer router (or a BGPmon chain) and allow BGPmon to provide data to at least one client.

### 2.5.1 Optional BGPmon Command Line Arguments

Nearly all configuration is done by loggin into BGPmon as discussed in Section 3. However, BGPmon has several optional command line arguments that can useful for some scenarios. Each of these optional values is discussed below.

- *-r recovery-port*: instructs BGPmon to allow administrator login on the specified port. If this option is not specified, BGPmon uses the port specified in the configuration file or port 50,000 if no configuration file is found. Section 3.2.1 describes how to set the login-listener port and save the setting in the configuration file.

  The *-r recovery-port* option is intended as a temporary bypass in case either 1) this is the first time BGPmon is running it needs to allow login on a port other than 50,000 or 2) the login port set in the configuration file is no longer valid and must be over-ridden. **The *-r recovery-port* option takes precedence over any login port found in Configuration File.**

- *-c filename*: provides the name of a configuration file to load. The configuration file provides essential information such as the peers to monitor, client access control, and so forth. If no configuration file is specified, bgpmon attempts to load a default configuration file name, bgpmon_config.txt. If the

configuration file is not found, BGPmon simply waits for an administrator to login and configure BGPmon.

Most users will not need to use the *-c filename* option. Unless you specified otherwise, saving a configuration creates the file bgpmon_config.txt and this file is loaded by default when BGPmon restarts. If you plan to have only one BGPmon configuration file, the default configuration file name of bgpmon_config.txt is *strongly recommended*. A site with multiple, distinct configurations may wish to use other file names and can specify which of the multiple configuration files to load using the *-c filename* option.

- *-i* : sets bgpmon to run in an interactive mode and sends all messages to stdout. Interactive mode is useful for debugging, especially in port BGPmon to new operating systems or troubleshooting installation problems. If this option is not specified, BGPmon writes all messages using the syslog facility.

- *-s* : sets bgpmon to run in an syslog mode and sends all messages to the syslog facility. Syslog mode is recommended and syslog settings can be used to direct BGPmon output to specific file, control the level of output, and so forth.

  The *-i* and *-s* options are mutually exclusive and the program exits with an error if both are specified. If neither option is specified, BGPmon logs messages in syslog mode. A site administrator may modify the source code in order to change the default setting to interactive mode, see [2] for instructions on modifying the default settings.

- *-l loglevel* option specifies the log level and uses the standard syslog values as follows. Emergencies, Alerts, Critical Errors, and Errors are levels 0 to 3 (respectively). These messages are always logged regardless of the log level setting. Warnings, Notices, Information, and Debug output are levels 4 to 7 (respectively). Setting $loglevel = L$ will log all messages at and below the $L$. For example, a log level of 4 will display Alerts, Critical Errors, Errors, and Warnings, but will not display Notices, Information, or Debugging output. If the *-l loglevel* option is not specified, a default logvalue of level 4 (Warning) is used. A site administrator may modify the source code in order to change the default logvalue, see [2] for instructions on modifying the default settings.

- *-l loglevel* option specifies the syslog Facility. This option has no effect if messages are written to standard output (e.g. if *-i* was specified). If the *-f facility* option is not specified, a default syslog facility of USER is used. A site administrator may modify the source code in order to change the default logvalue, see [2] for instructions on modifying the default settings.

# 3 Configuring BGPmon

To configure BGPmon a user must first login to the Command Line Interface. Once connected, a user is initially set to *guest mode* which allows them to view statistics, show routing tables, and generally view (but not change) configuration parameters. In order to change the configuration settings, a user must switch to *privileged mode*. In this mode a user can perform BGPmon configuration actions such as adding, deleting, or modifying BGP peers and chains, disabling clients, and setting access control policies. At any time, a user in *privileged mode*, can save the current BGPmon configuration so it will be loaded the next time BGPmon starts. **Any configuration changes are stored in memory and will be lost if BGPmon restarts. To make changes permanent, the administrator must save the BGPmon configuration.**

This section describes how to configure BGPmon, beginning with logging into BGPmon and proceeding through the steps to configure future login access, enable clients, configure chains, create peers, and finally set optional parameters. A first time administrator should *read each subsection in order* and follow the configuration steps in that section. An experienced administrator may want to skip directly to the relevant subsection. A complete command reference is also available beginning in Appendix A.

## 3.1 Logging Into BGPmon

To log into the BGPmon server first ensure that it's running. Instructions for starting the server can be found in Section 2.5. If BGPmon wasn't previously configured then it will be listening on the loopback address and port 50,000 for incoming connections. From the same machine used to start BGPmon, simply enter:

```
telnet loopback 50,000
```

Some operating system and/or DNS setups will not recognize the name *loopback*. If the name loopback is not recognized, you can use the loopback IP address instead of the name *loopback*. For IPv4, the loopback address is 127.0.0.1 and for IPv6, the loopback address is 0:0:0:0:0:0:0:1.

After connecting to BGPmon, you will be prompted for a guest password. By default, the guest password is 'BGPmon'. Initially, a user is connected in *guest mode* and must change to *privileged mode* to alter the server's configuration settings. To enter *privileged mode*, type:

```
enable
```

You will then be prompted for the enable password, which is also 'BGPmon' by default. From *privileged mode* you need to enter *configure mode* to gain access to all the configuration commands. To enter this mode type:

```
configure
```

Now you are ready to configure BGPmon. It is recommended you begin by changing the default passwords and configuring future login access addresses, ports, and restrictions as discussed in Section 3.2.

### 3.1.1 Site Specific Login Instructions

The following steps are used login to BGPmon:

```
telnet <address> <port>
<enter guest password>
enable
<enter privileged password>
configure
```

The address, port, and passwords are all settings that depend on how BGPmon was configured. By default, the address is either 127.0.0.1 (IPv4 machines) or 0:0:0:0:0:0:0:1 (IPv6 machines), the port is 50,000, and both passwords are 'BGPmon'. The local BGPmon administrator may also have created *access control lists* that restrict which machines can login to BGPmon. **If the default values do not work, you need to obtain the proper values from the local BGPmon administrator.**

If you are the BGPmon administrator and have lost the login settings, Section 3.1.2 describes how to reset these values without losing the other configuration information.

### 3.1.2 BGPmon Login Recovery

Due to configuration errors and/or misplaced information, an administrator may be unable to login to BGPmon. For example, the administrator may have lost the guest password or, in a more complex scenario, setup an overly aggressive access control list to the BGPmon Command Line Interface. In these cases, the administrator has four options for recovery.

The first option is to use the *-r recovery-port* parameter described in section 2.5. This parameter will override the login-listener's port so an administrator can still use the Command Line Interface if there is a conflict between BGPmon and another application trying to use the same port. However, the recovery-port will not allow an administrator to bypass the ACL specified for the login-listener. If this is the problem then

one of the following options for recovery must be used. The second option can be used in cases where the configuration is simple or very close to the default settings. Simply delete the active configuration, which is stored in 'bgpmon_config.txt', and let BGPmon start with the default settings. Then make any necessary changes to the BGPmon configuration and save the settings.

There are many times where starting over or simply bypassing the default port won't be a viable option and for these cases there are several other options. The first is trying to recover an old configuration. Any configuration changes that are saved will be written into 'bgpmon_config.txt'. Every time a new configuration is saved the old configuration is archived in the same directory with the time and date preceding the file name. An example archived configuration file is '1041_2252009_bgpmon_config.txt', which was created at 10:41am on 02/25/2009. To find a suitable recovery point, start by backing up the current configuration then copying the archived configuration over the current configuration. Hopefully within a few tries a good recovery point can be identified.

The final option is to manually edit the active configuration with a text editor. It's easy to manually edit the active configuration for BGPmon since all settings are stored as XML in 'bgpmon_config.txt'. By comparing the last known good configuration and the current configuration all changes between these versions can be identified then reapplied to the current configuration. Just be careful to make changes slowly until the unwanted change to BGPmon is identified and corrected.

## 3.2 Configuring Login Access To BGPmon

Login access is controlled in BGPmon by the login-listener. This module will bind to an address and port then listen for incoming Command Line Interface connections. When a connection is established it will check to see if the address of the new connection is allowed or disallowed based on rules setup in the Access Control List. Use the following commands to see what the current settings of the login-listener are:

Address:

```
show login-listener address
```

Port:

```
show login-listener port
```

Access Control List:

```
show login-listener acl
```

### 3.2.1 Configuring the login-listener

There are three main components that can be configured for login-listener. First is the address that the login-listener will attempt to use when BGPmon starts. To change the login-listener's address simple type the following command:

```
login-listener address new-address
```

BGPmon will check the *new-address* to make sure that it's a valid IP address and is available on the local machine. If the address isn't valid then a warning will be returned and nothing is set. Also, any loopback address in Appendix B can be used.

The next major component to configure is the port. To change the port, type the following command:

```
login-listener port new-port
```

BGPmon will check to make sure the *new-port* is a valid port number but will not check to see if the port can be bound. So, if the port for the login-listener is set to an unavailable port and BGPmon is restarted

then the Command Line Interface will not be available upon restart. If this happens see section 3.1.2 about recovering.

The final component for the login-listener is the Access Control List, which is the list that controls which addresses are allowed to connect or not connect. To set the active ACL, type the following:

```
login-listener acl acl-name
```

BGPmon will check to make sure the *acl-name* is valid within BGPmon before setting it. Refer to section 3.4 to learn about configuring Access Control Lists.

## 3.3 Configuring Client Access

Similar to the login-listener, client access is controlled in BGPmon by the client-listener. This module will bind to an address and port then listen for incoming connections. When a connection is established it will check to see if the address of the new connection is allowed or disallowed based on rules setup in the Access Control List. Use the following commands to see what the current settings of the client-listener are:

Address:
```
show client-listener address
```

Port:
```
show client-listener port
```

Access Control List:
```
show client-listener acl
```

Status:
```
show client-listener status
```

### 3.3.1 Configuring the client-listener

There are four main components that can be configured for the client-listener. Any change made to these components will result in the client-listener stopping then starting, if necessary, with the new values. The first component is the address that the client-listener will use. To change the address type the following command:

```
client-listener address new-address
```

BGPmon will check the *new-address* to make sure that it's a valid IP address and is available on the local machine. Also, any loopback address in Appendix B can be used.

The next major component to configure is the port. To change the port, type the following command:

```
client-listener port new-port
```

BGPmon will check to make sure the *new-port* is a valid port number but will not check to see if the port can be bound until the client-listener stops then attempts to start again.

The next component for the client-listener is the Access Control List, which is the list that controls which addresses are allowed to connect or not connect. To set the active ACL, type the following:

```
client-listener acl acl-name
```

BGPmon will check to make sure the *acl-name* is valid within BGPmon before setting it. Refer to section 3.4 to learn about configuring Access Control Lists.

The final component is the status, which can be set to either *enabled* or *disabled*. Use the following commands to change the status:

```
client-listener acl {enable | disable}
```

## 3.4 Access Control Lists

Access Controls Lists are used within BGPmon to control which addresses are or aren't allowed to connect. BGPmon initially comes with two ACLs: 'denyall' which denies all traffic and 'permitall' which permits all traffic. To see a list of ACLs that are available, type the following command:

```
show acl [acl_name]
```

The optional parameter, acl_name, can be included to limit the results to a specific ACL. When an ACL is returned it will list the name of the ACL and all the associated rules. Below is example output:

```
ACL name:denyall
index address               mask                allowed
0     any                   any                 deny

ACL name:permitall
index address               mask                allowed
0     any                   any                 permit

ACL name:testacl
index address               mask                allowed
0     192.168.0.0           255.255.0.0         permit
1     any                   any                 deny
```

### 3.4.1 ACL inverse mask logic

Access Control Lists in BGPmon are designed to give administrators a way of controlling which addresses are allowed to connect and which are not. Each ACL is made up of a series of rules and each rule is made up of three components. The first two components, address and mask, are used to determine whether the rule is applicable to the incoming address. The third component, allowed, tells BGPmon whether to permit or deny applicable addresses.

Below is an example of how BGPmon determines if a rule is applicable to incoming address. The first step is to OR the rule's address and mask together then OR the incoming address and mask together. Subtracting these two results will indicate whether a rules is applied. A zero value means that the rule is applied and a non-zero value means that the rule does not apply.

| **Example 1** | | |
|---|---|---|
| | **String** | **Binary** |
| Incoming address | 10.1.1.255 | 00001010 00000001 00000001 11111111 |
| Rule address | 10.1.1.1 | 00001010 00000001 00000001 00000001 |
| Rule mask | 0.0.255.255 | 00000000 00000000 11111111 11111111 |
| | | |
| incoming address \| mask | | 00001010 00000001 11111111 11111111 |
| rule address \| mask | | 00001010 00000001 11111111 11111111 |
| **zero difference - rule should be applied** | | 00000000 00000000 00000000 00000000 |

| Example 2 | | |
|---|---|---|
| | **String** | **Binary** |
| Incoming address | 10.1.255.255 | 00001010 00000001 11111111 11111111 |
| Rule address | 10.1.1.1 | 00001010 00000001 00000001 00000001 |
| Rule mask | 0.0.0.255 | 00000000 00000000 00000000 11111111 |
| | | |
| incoming address \| mask | | 00001010 00000001 11111111 11111111 |
| rule address \| mask | | 00001010 00000001 00000001 11111111 |
| **non-zero difference - rule should not be applied** | | 00000000 00000000 11111110 00000000 |

### 3.4.2 Creating and Editing ACLs

To either create or edit an ACL type the following:

```
acl acl_name
```

This command will attempt to open the specified ACL. If it's not found then a new ACL is created. Within the ACL edit mode there are several commands that allow the user to maintain an ACL.

The first set of commands are used to create 'permit' or 'deny' rules.

```
permit any [rule_index]
permit address mask [rule_index]
deny any [rule_index]
deny address mask [rule_index]
```

In each of these commands either the 'permit' or 'deny' keyword is specified then followed by a series rules that indicates the range of addresses the rule applies to. The parameter *address* is the IP address used in the rule. The parameter *mask* is used to specify which bits in the address are significant and which should be ignored. If a rule should be applied to all addresses then use the keyword 'any' instead of the address/mask pair. Finally the optional parameter *rule_index*, if set, will be used to specify where in the list the rule should be inserted. When a rule is inserted at a rule_index, then all rules that follow will be incremented by one.

To remove a rule from the list use the following command:

```
no rule_index
```

When a rule is removed from the list all other rules in the list will be re-indexed while maintaining their relative ordering.

### 3.4.3 Deleting ACLs

To delete an ACL and all associated rules use the following command:

```
no acl acl_name
```

It is important to note that when an ACL is removed, any modules using that ACL will be set to the default behavior, which is to deny all traffic.

## 3.5 Configuring Chains

Chaining is used in BGPmon to connect multiple versions of BGPmon together so XML messages from one BGPmon instance can be sent to another BGPmon instance.

### 3.5.1 Creating and updating a chain

To create a chain use the following command:

`chain` *address* [*port*] [enable | disable] [retry:*retry_interval*]

The only required parameter is address and any parameter which is not specified will have a default value assigned to it. The defaults are as follows: port is 50,001, chain status is enabled, and retry interval is 60. Once a chain has been created, the same command can be used to update settings for that chain.

For example, assume a chain was created with the following command:

`chain` *192.168.1.1*

To set the status of this chain to disabled and retry interval to 10, use the following command:

`chain` *192.168.1.1* disable retry:*10*

One important thing to remember about creating chains is that the address and port uniquely identify a chain. So, assume a chain is created with the following command:

`chain` *192.168.1.1*

Now, assume this command is run:

`chain` *192.168.1.1 50002*

The second command will create a new chain at *192.168.1.1* on port *50002* while the first chain will remain at *192.168.1.1* on port *50001*. **Once a chain has been created its address and port cannot updated.** Any command attempting to do this will create another chain.

### 3.5.2 Delete a chain

A chain can be deleted with the following command:

`no chain` *address* [*port*]

A port doesn't need to be specified if the default port was used when creating the chain. When this command is issued BGPmon will set the chain to disabled then then mark the chain to be deleted.

## 3.6 Configuring Peers and Peer Groups

A Peer is the connection that BGPmon maintains with a router and every Peer has its own set of configurable parameters. Peer Groups are a way of sharing configurations between sets of Peers .

The first step in configuring a peer is to enter the router configuration mode. To do this, enter the following command from the configuration mode prompt:

`router bgp` *AS_Number*

The *AS_Number* entered will be used as the local AS number for any peers that are created. Every command associated with a peer or peer-group in this mode follows the same structure: a base command used to identify the peer or peer-group followed by a command to modify that peer or peer-group. The base

of the command is as follows:

```
neighbor {address | peer-group} [port neighbor-port]
```

If a valid IP address is specified for the *address* then the command will refer to a peer. The optional *neighbor-port* parameter can then be used to specify a custom port, otherwise the default port 179 will be used. If the *address* is not a valid IP address then the command will refer to a peer-group.

### 3.6.1 Creating Peers

To create a peer run any command where the base command has a valid address. If the peer does not exist then BGPmon will attempt to create the peer. Example:

```
neighbor 192.168.1.1 [port 4626] announce ipv4 unicast
```

If no pre-existing peers are configured, then this command would create a peer with an address of *192.168.1.1* and a port of *4626*.

### 3.6.2 Creating Peer-groups

To create a peer-group the following command is used, where *peer-group* will be the name of the peer-group and not a valid IP address:

```
neighbor peer-group peer-group
```

Once a peer-group has been created then peers can be assigned to the group with the following command:

```
neighbor address [port neighbor_port] peer-group peer_group_name
```

A peer can be moved to another peer-group with the same command and use the following command to remove a peer from a peer-group:

```
neighbor address [port neighbor_port] peer-group peer_group_name
```

### 3.6.3 Configuring Parameters

Within a Peer or Peer-group there are many different parameters that can be configured. The first group are the commands that configure parameters for the *remote* and *local* BGP versions. These commands are as follows:

```
neighbor {address | peer-group} [port neighbor port] {remote | local} as [as_number]
neighbor {address | peer-group} [port neighbor port] {remote | local} bgpid [bgpid]
neighbor {address | peer-group} [port neighbor port] {remote | local} bgp-version [bgp_version]
neighbor {address | peer-group} [port neighbor port] {remote | local} hold-time [hold_time]
```

The second group of commands are the labeling commands. These commands control how messages are labeled in each peer:

```
neighbor {address | peer-group} [port neighbor port] label-action NoAction
neighbor {address | peer-group} [port neighbor port] label-action Label
```

```
neighbor {address | peer-group} [port neighbor port] label-action StoreRibOnly
```

The final group of commands are commands used to control the actual peer. These commands will trigger a route-refresh, enable and disable a peer, and set the md5 encryption for a password.

```
neighbor {address | peer-group} [port neighbor port] route-refresh
neighbor {address | peer-group} [port neighbor port] enable
neighbor {address | peer-group} [port neighbor port] disable
neighbor {address | peer-group} [port neighbor port] md5 [md5_password]
```

### 3.6.4 Configuring Capabilities

For every peer there are a set of *announce* and *receive* capabilities that can be configured:
Announce capabilities:

```
neighbor {address | peer-group} [port neighbor-port] announce code length value
neighbor {address | peer-group} [port neighbor-port] announce {ipv4 | ipv6} {unicast | multicast}
```

Receive capabilities:

```
neighbor {address | peer-group} [port neighbor-port] receive {require | refuse | allow} code
length value
neighbor {address | peer-group} [port neighbor-port] receive {require | refuse | allow} {ipv4
| ipv6} {multicast | unicast}
```

### 3.6.5 Deleting a peer

To delete a peer, use the following command:

```
no neighbor address [port port]
```

To delete a peer-group, use the following command:

```
no neighbor peer-group
```

### 3.6.6 Reseting a peer's connection

Reseting a peer will close the peer's connection then attempt to reopen it. Any new settings that have been applied to the peer will be applied when the peer starts again. To reset a connection to a peer use the following command:

```
clear neighbor address [port port]
```

# A Command Line Interface Reference

**Privilege Level:** Access Control List Configuration

Mode commands:

**exit** - Sets the security level of the current user back to Configure.

**end** - Sets the security level of the current user back to Privileged.

Rule commands:

**no *rule_number*** - Deletes an ACL rule based on the rule number.

**no permit *rule_number*** - Deletes a permit ACL rule based on the rule number.

**no deny *rule_number*** - Deletes a deny ACL rule based on the rule number.

**no label *rule_number*** - Deletes a deny ACL rule based on the rule number.

**no ribonly *rule_number*** - Deletes a deny ACL rule based on the rule number.

**permit any** [***rule_index***] - Creates a rule that permits any address. If this rule is applied to the Quagga module then only updates are allowed from the address.

**deny any** [***rule_index***] - Creates a rule that denies any address.

**label any** [***rule_index***] - Creates a rule that allows only LABEL messages. This rule should only be applied to the Quagga module.

**ribonly any** [***rule_index***] - Creates a rule that allows only RIB messages. This rule should only be applied to the Quagga module.

**permit *address mask*** [***rule_index***] - Creates a rule that permits any address which matches the address and mask combination specified in the rule. If this rule is applied to the Quagga module then only updates are allowed from the address.

**deny *address mask*** [***rule_index***] - Creates a rule that denies any address which matches the address and mask combination specified in the rule.

**label *address mask*** [***rule_index***] - Creates a rule that allows LABEL messages from an address which matches the address and mask combination specified in the rule.

**ribonly *address mask*** [***rule_index***] - Creates a rule that allows only RIB messages any address which matches the address and mask combination specified in the rule.

**Privilege Level:** Router Configuration

Announce and Receive commands:

**neighbor {*peer-group* | *address*} [port *neighbor_port*] announce *code length value*** - Adds a custom announce capability to the neighbor or peer-group specified.

**neighbor {*peer-group* | *address*} [port *neighbor_port*] announce {ipv4 | ipv6} {unicast | multicast}** - Adds a predefined announce capability to the neighbor or peer-group specified.

**neighbor {*peer-group* | *address*} [port *neighbor_port*] receive {require | refuse | allow} *code length value*** - Adds a custom receive capability to the neighbor or peer-group specified.

**neighbor {*peer-group* | *address*} [port *neighbor_port*] receive {require | refuse | allow} {ipv4 | ipv6} {multicast | unicast}** - Adds a predefined receive capability to the neighbor or peer-group specified.

Creating a peer-group:

**neighbor *peer-group* peer-group** - Creates a peer-group.

**neighbor *address* [port *neighbor_port*] peer-group *peer_group_name*** - Assigns a peer to a peer-group.

Mode commands:

**exit** - Sets the security level of the current user back to Configure.

**end** - Sets the security level of the current user back to Privileged.

Remote and Local commands:

**neighbor {*peer-group* | *address*} [port *neighbor_port*] {remote | local} as [*as_number*]** - Sets either the remote or local AS number for a peer or peer-group.

**neighbor {*peer-group* | *address*} [port *neighbor_port*] {remote | local} bgpid [*bgpid*]** - Sets either the remote or local BGPID for a peer or peer-group.

**neighbor {*peer-group* | *address*} [port *neighbor_port*] {remote | local} bgp-version [*bgp_version*]** - Sets either the remote or local BGP version for a peer or peer-group.

**neighbor {*peer-group* | *address*} [port *neighbor_port*] {remote | local} hold-time [*hold_time*]** - Sets either the remote or local hold-time for a peer or peer-group.

Upadating a peer or peer-group:

**no neighbor {*peer-group* | *address*}** - Deletes a neighbor or peer-group.

**clear neighbor *address* [port *neighbor_port*]** - Resets the connection with the neighbor.

**neighbor {*peer-group* | *address*} [port *neighbor_port*] enable** - Enables a peer's connection.

**neighbor {*peer-group* | *address*} [port *neighbor_port*] label-action NoAction** - Sets the labelling action for a peer-group or peer to NoAction, stopping all labelling.

**neighbor {*peer-group* | *address*} [port *neighbor_port*] label-action Label** - Sets the labelling action for a peer-group or peer to Label, starting all labelling.

**neighbor {*peer-group* | *address*} [port *neighbor_port*] label-action StoreRibOnly** - Sets the labelling action for a peer-group or peer to StoreRibOnly, stopping labelling but storing the RIB-IN table.

**neighbor {*peer-group* | *address*} [port *neighbor_port*] route-refresh** - Triggers a route-refresh for that peer from the BGPmon RIBIN table.

**neighbor {*peer-group* | *address*} [port *neighbor_port*] md5 [*md5_password*]** - Sets the MD5 password used in authenticating with a peer.

**neighbor {*peer-group* | *address*} [port *neighbor_port*] disable** - Disables a peer's connection.

**Privilege Level:** Configuration

ACL commands:

**no acl *acl name*** - Deletes an Access Control List.

**acl *acl_name*** - Creates a new ACL or opens an existing ACL then changes the security level of the user to ACL Configuration.

Chain commands:

**chain *address*** [***port***] [**enable** | **disable**] [**retry:*retry_interval***] - blah

**no chain *address*** [***port***] - Deletes a Chain.

Client commands:

**client-listener update address *local-address*** - Sets the address that the client update listener will listen on.

**client-listener update port *port*** - Sets the port that the client update module will listen on for new client connections.

**client-listener update acl *acl_name*** - Sets the ACL which will be applied to the clients update modules.

**client-listener rib address *local-address*** - Sets the address that the client rib listener will listen on.

**client-listener rib port *port*** - Sets the port that the client rib module will listen on for new client connections.

**client-listener rib acl *acl_name*** - Sets the ACL which will be applied to the clients rib modules.

**client-listener {enable** | **disable}** - Enables or disables the listener, both update and rib, for the client module.

Login commands:

**login-listener address *local-address*** - Sets the address that the login-listener will listen on.

**login-listener port *port*** - Sets the port that the login-listener will listen on.

**login-listener acl *acl_name*** - Sets the ACL that will be applied to the login-listener.

Mode commands:

**exit** - Sets the security level of the current user to Privileged.

**end** - Sets the security level of the current user to Privileged.

Neighbor commands:

**router bgp *monitor_AS_number*** - Allows the user to enter Router Configuration mode where peers and peer groups can be modified.

Periodic commands:

**periodic route-refresh *route_refresh*** -

**periodic status-message *status_message*** -

Queue commands:

**queue pacingOnThresh *pacingOnThresh*** - Modifies the pacing-on threshold value for queues.

**queue pacingOffThresh *pacingOffThresh*** - Modifies the pacing-off threshold value for queues.

**queue alpha *alpha*** - Modifies the alpha value for queues.

**queue minWritesLimit *minimum_write_limit*** - Modifies the minimum write limit for queues.

**queue pacingInterval *pacing_interval*** - Modifies the maximum write limit for queues.

| **Privilege Level:** Privileged |
| --- |
| Client commands: |
| **kill client *client_id*** - Disconnects the client specified by the client_id. |
| Configuration commands: |
| **copy running-config *filename*** - Copies the running configuration to a filename specified.<br>**copy running-config startup-config** - Copies the running configuration to the startup configuration.<br>**shutdown** - Shuts down the bgpmon server. |
| Mode commands: |
| **configure** - Sets the security level of the current user to Configure.<br>**disable** - Sets the security level of the current user back to Guest.<br>**exit** - Exits the BGPmon Command Line Interface. |

| **Privilege Level:** Guest |
|---|
| ACL commands: |
| **show acl** [***acl_name***] - Shows the details of an Access Control List. |
| Chain commands: |
| **show chains** [***chain_id***] - Shows information about one or all currently configured chains. |
| Client commands: |
| **show clients** - Shows the list of users actively connected on the Client module.<br>**show client-listener status** - Shows the status of the Client-listener.<br>**show client-listener update port** - Shows the current port for the client-listener's update channel.<br>**show client-listener update address** - Shows the current address for the client-listener's update channel.<br>**show client-listener update acl** - Shows the Access Control List assigned to the client-listener's update channel.<br>**show client-listener rib port** - Shows the current port for the client-listener's rib channel.<br>**show client-listener rib address** - Shows the current address for the client-listener's rib channel.<br>**show client-listener rib acl** - Shows the Access Control List assigned to the client-listener's rib channel. |
| Configuration commands: |
| **show running** - Shows running configuration for the instance of BGPmon. |
| Login commands: |
| **show login-listener port** - Shows the current port for the login-listener.<br>**show login-listener address** - Shows the current address for the login-listener.<br>**show login-listener acl** - Shows the Access Control List assigned to the Command Line Interface module. |
| Mode commands: |
| **enable** - Sets the security level of the current user to Privileged.<br>**exit** - Exits the BGPmon Command Line Interface. |
| Neighbor commands: |
| **show bgp neighbor** [***neighbor_address***] **port** [***neighbor_port***] - Shows information about a configured peer. |
| Periodic commands: |
| **show periodic route-refresh** - Shows the route-refresh interval.<br>**show periodic route-refresh status** - Shows the status, enabled or disabled, of route-refresh.<br>**show periodic status-message** - Shows the status message interval. |
| Quagga commands: |
| **show quagga-listener status** - ****Shows the status of the quagga-listener.<br>**show quagga-listener port** - ****Shows the current port for the quagga-listener.<br>**show quagga-listener address** - ****Shows the current address for the quagga-listener.<br>**show quagga-listener acl** - Shows the Access Control List assigned to the quagga-listener. |

| **Privilege Level:** Guest |
| --- |
| Queue commands: |
| **show queue** - Shows general information about the Queues.<br><br>**show queue PeerQueue** - Shows general information about Queues and information specific to the Peer Queue.<br><br>**show queue LabelQueue** - Shows general information about Queues and information specific to the Label Queue.<br><br>**show queue XMLUQueue** - Shows general information about Queues and information specific to the XML Queue that contains all update messages.<br><br>**show queue XMLRQueue** - Shows general information about Queues and information specific to the XML Queue that contains all the RIB table messages. |

# B IP Address reference

| **Shortcut** | **Version** | **Actual** |
|---|---|---|
| ipv4any | IPv4 | any address |
| ipv6any | IPv6 | any address |
| ipv4loopback | IPv4 | loopback address = 127.0.0.1 |
| ipv6loopback | IPv6 | loopback address = ::1 |

# References


[1] D. Matthews, N. Parrish, H. Yan, , and D. Massey. BGPmon: A real-time, scalable, extensible monitoring system. *Proceedings of the ACM SIGCOMM Internet Measurement Confernce (IMC)*, 2008.

[2] D. Matthews, H. Yan, , and D. Massey. BGPmon Implementation and Technical Specification, 2008.

[3] M. Welsh, D. Culler, and E. Brewer. Seda: an architecture for well-conditioned, scalable internet services. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 230–243, New York, NY, USA, 2001. ACM.