Policy Traffic Switch SandScript Configuration Guide, Release 6.30

05-00217-D01
2013-12-20

The most current version of this document is available on the Sandvine Customer Support web site at *https://support.sandvine.com*.

Sandvine is committed to ensuring the accuracy of our documentation and to continuous improvement. If you encounter errors or omissions in this user guide, or have comments, questions, or ideas, we welcome your feedback. Please send your comments to Sandvine via email at *https://support.sandvine.com*.

Contacting Sandvine

To view the latest Sandvine documentation or to contact Sandvine Customer Support, register for an account at *https://support.sandvine.com*.

For a list of Sandvine Sales and Support offices, see *http://www.sandvine.com/about_us/contact.asp*.

Related documentation is available from the Sandvine Customer Support web site.

All documents are in PDF format, which you can open, read, or print using Adobe® Acrobat® Reader®. You can obtain a free copy of this software from the Adobe® web site.

Document	**Part Number**
PTS Release Notes	05-00214-E01
PTS Hardware Installation Guide	05-00185-A11
PTS Software Installation Guide	05-00245-B01
PTS Network Configuration Guide	05-00192-D01
PTS SandScript Configuration Guide	05-00217-D01
PTS SandScript Reference Guide	05-00069-C01
PTS Alarm Reference Guide	05-00262-A01
PTS CLI Reference Guide	05-00263-A01

Contents

1

SandScript Configuration

1.1 Introduction to SandScript

SandScript is an event-based policy definition language that lets Communications Service Providers (CSPs) fully leverage the Sandvine policy engine with programmatic flexibility.

Sandvine's policy engine can be thought of as a "green box" into which conditions and subscriber entitlements flow, and out of which charging updates, management actions, and signals emerge. It evaluates SandScript logic on the lowest element in the network that has access to the requisite information. This approach maximizes scalability by reducing centralization, and maximizes performance by removing unnecessary communication between elements, thereby avoiding conditions of signaling overload.

These components work in concert and leverage the policy engine:

- The Policy Traffic Switch (PTS) identifies and measures traffic, makes local SandScript decisions and implements enforcement actions (fulfilling the functions of a PCEF).
- The Service Delivery Engine (SDE) is a SandScript decision and enforcement element that, like a PCRF, works together with the PTS to provide unified network policy control within multi-vendor and multi-access networks.
- The Subscriber Policy Broker (SPB) acts as a subscriber profile repository (SPR) and provides long-term storage for comprehensive business intelligence insight.

SandScript can be used to:

- Identify network conditions that are approaching their limit for a subscriber, application, or location.
- Evaluate business logic rules in real-time.
- Enforce SandScript actions that affect network conditions and subscriber experiences.
- Define measurements and collect statistics for business intelligence insight into network traffic, subscriber usage, application quality of experience, and more.
- Configure the function of the PTS and SDE elements under specific circumstances.

1.1.1 Rules

The SandScript policy language is based on a collection of rules composed of conditions and actions.

Rules may be applicable in one or more contexts, such as subscriber context or Diameter context. Whenever the conditions of a rule evaluate to true, the actions of that rule are performed.

Conditions select a subset of traffic or events of a specific type or having a specific attribute. Examples of conditions include:

Conditions select events of a specific type or having a specific attribute. Examples of conditions include:

- An application protocol such as BitTorrent or YouTube.
- A transport protocol such as TCP, UDP, GRE, or ICMP.
- An interface on the physical element.
- Subscriber properties such as a tier (for example gold tier).
- A transfer direction, either upload or download.
- A server within the network or external to the network.
- A time of day such as from 9 AM to 5 PM.
- A measurement such as peer-to-peer traffic exceeding 300 MB.
- Events generated by protocol subsystems such as DHCP, RADIUS, or Diameter.

Actions are performed on the subset of traffic selected by the condition. An action is applied while the condition remains true. Some examples of actions are:

Actions are performed on the subset of events selected by the condition. An action is applied while the condition remains true. Some examples of actions are:

- Shape traffic.
- Tee traffic to a particular destination.
- Count or measure a particular statistic of the traffic.
- Set a subscriber attribute.
- Modify state in a SandScript table.
- Send a RADIUS or Diameter message.
- Generate a logging record.
- Execute a system command.

Conditions and actions are used together: determine what action you want performed on the traffic, then create the condition to filter the traffic on which to perform the action.

1.1.2 SandScript Process

The Policy Traffic Switch (PTS) is a key component of Sandvine's network policy control platform, enabling the real-time application of business logic and policy enforcement on data traffic. The PTS is powered by the Sandvine policy engine and a high-performance packet processing operating system executing on purpose-built, carrier-grade hardware.

The method of deploying a PTS element into a network determines the type of actions that it can perform:

- Offline (passive) deployment – The element is not deployed in the direct path of any subscriber data; instead, it is connected to either a mirror port (such as a Switched Port Analyzer, or SPAN) or via some sort of media splitter, such as an optical tap. In this method of deployment, the actions that modify outbound traffic, such as shape or block, have no effect.
- Inline (active) deployment – The element is deployed in the direct path of the subscribers' data. The full suite of actions can be applied.

1.2 SandScript Configuration Files

Use Control Center's **Power Edit** or **Quick Edit** to modify and manage these configuration files:

File Location	Description	Platform
`/usr/local/sandvine/etc/subnets.txt`	Defines a text-based map of the network labeling internal versus external IP subnets. Internal and external designations are relative to the PTS and its interfaces.	PTS
`/usr/local/sandvine/etc/subnets.xml`	Defines an XML map of the network identifying network and policy classes. It associates the names of classes with groups of IP addresses or subnets and assigns attributes to the network and policy classes.	SDE
`/usr/local/sandvine/etc/policy.conf`	Contains SandScript rules and actions that act on traffic or events.	PTS, SDE, and SPB

1.2.1 Control Center

Control Center is Sandvine's unified policy and operations management graphical user interface, providing a single mechanism for monitoring operational information, editing network policies, configuring elements, and deploying network policy control solutions.

Control Center lets communications service providers (CSPs) centrally create and deploy service policies for the entire network in response to new opportunities or trends. This simplifies all aspects of Sandvine operations management, delivering real-time information and granular control.

Control Center lets you safely configure SandScript policy in isolation from the physical devices in order to control when the elements enforce new SandScript behavior. Control Center is also a repository of configuration information for the elements (PTS, SDE, and SPB). The elements are responsible for managing real-time data and enforcing SandScript. Control Center manages all SandScript and configuration changes in its database. The migration of SandScript and configuration to the elements is called a deployment. Deployment is not automatic; you must start it manually.

1.2.1.1 Online Help

For further information about Control Center, see the online help. Control Center's policy creation wizards contain help content and tool tips are available when you hover your mouse over a GUI button.

1.3 SandScript File

A SandScript policy file is a text file consisting of definitions and rules.

Definitions in the SandScript file create complex constructs to be used in later definitions and rules. Rules consist of conditions and actions. A SandScript group can be made of rules and nested groups. A SandScript file implements and stores these SandScript components. The main SandScript file is `policy.conf`, but it can include several other SandScript policy files. The main components in a SandScript file are:

- Definition – Defines an attribute by giving it a name and a type.
- Rule – Describes the logic and actions of the SandScript policy.
- Include – Adds the rules from other SandScript files to this SandScript file.

1.3.1 Creating a SandScript File

The policy.conf file is the only SandScript file that is loaded onto the PTS or the SDE policy engine.

A sample SandScript file is available on each PTS or SDE element at `/usr/local/sandvine/etc/policy.conf.sample`. You can copy, then edit this sample file.

1. As an administrative user, connect to the element.
2. Copy the `/usr/local/sandvine/etc/policy.conf.sample` file to `/usr/local/sandvine/etc/policy.conf`.
3. Edit `/usr/local/sandvine/etc/policy.conf` as appropriate.

   Use Control Center or the `edit policy` CLI command.
4. If the cluster is locally configured, copy the `policy.conf` file to each element in the cluster. Or if centralized configuration is implemented for the cluster, update the file on the central PTS, then update the remote elements.

5. To reload the configuration files on each PTS element, run `reload`.

6. To reload the configuration files on each SDE element, run `svreload`.

1.4 The subnets.txt File

Each PTS element must have a subnets.txt file which identifies network and policy classes.

The subnets.txt file is a text-based map of the network that associates the names of classes with groups of Internet Protocol (IP) addresses or subnets, and assigns attributes to the network and policy classes. Use the `edit policy subnets` CLI command to open the subnet.txt file for edit. The full path is `/usr/local/sandvine/etc/subnets.txt`.

The subnets.txt file allows names to be associated with groups of IP addresses or subnets. These named groups are:

- Network classes – Define how demographic statistics are collected and can be used to create traffic policies in `policy.conf`.
- Policy classes – Can only be used to create traffic policies and have no effect on demographic statistics collection.

The subnets.txt syntax is:

```
#comments are preceded by the # symbol
className cost type
{subnet{site} | AS }

#blank lines are allowed
className cost type
{subnet{site} | AS}
{subnet{site} | AS}
{subnet{site} | AS}

#policy classes are preceded by the POLICY keyword
POLICY <policyClassName>
{subnet{site} | AS}
{subnet{site} | AS}
```

Where:

- className – Specifies the name of the network or policy class for the associated subnets.
- cost – Sets the cost associated with traffic to/from the network. Cost is evaluated as relative to other network classes defined in subnets.txt, such that 10 is less than 20. The internet (default) class should be the highest.
- type – Indicates the type of the network class:
  - Internal – Subnets are internal to the network.
  - External – Subnets are external to the network.
  - Peer
- subnet – Subnets to associate with the network or policy class.

  Malformed IP addresses, those that are padded with leading zeros or blank spaces, cause a reload error and are logged to `/var/log/svlog`.
- site – A site number that further qualifies a subnet in situations where subnets overlap. Specified in the format:

  `subnet; site: n`

  For example:

  `10.10.10.1/10; site: 1`

- AS – Autonomous System (AS) numbers to associate with the network or policy class. Uses Border Gateway Protocol (BGP) to populate network or policy classes.
- <policyClassName> – Name of the policy class used to apply differentiated policy to the associated subnets.

Note that:

- Longest prefix match is used to classify an IP address into exactly one network class, and zero or one policy class. If the same prefix is used in two classes, the result is not defined.
- Statistics are only logged to network classes, not policy classes.
- You can write policies to reference policy classes in the same manner as regular network classes.
- You can create multiple policy classes to a maximum of 10,000.
- A default policy class is not required. If no default policy class exists, some ranges of addresses may not have a policy class value.

The subnets described by this file should match those described in subnets.xml on the SDE. See the *SDE User Guide* for details.

Default subnets.txt File

The PTS includes a default subnets.txt file and a `subnets.txt.sample` file. The default subnets.txt contains this default cost class:

```
default 100 external
0.0.0.0/0
::/0
```

Example:
This is an example of an internal network class with a single subnet:

```
internal-1 10 internal
10.10.10.10/1
```

Example:
This example uses a session qualifier.

```
subs-in-site 10 internal
10.10.10.10/1; site:1
```

Example:
This example is of an internal network class with multiple subnets. Notice that the IPs in the previous example are also in 4.1.0.0/16 below, but since 4.1.0.0/24 is a smaller subnet, those IPs belong to internal-1.

```
internal-2 20 internal
10.10.10.10/1
10.10.10.10/2
10.10.10.10/3
10.10.10.10/4
1001:0:5e81:992f::/48
1001:0:4::/56
```

Example:
This example is of an internal network class that uses BGP to automatically discover subnets.

```
internal-bgp 20 internal
AS12345
AS54321:13
```

Example:
Here is an example of a peer network class with multiple subnets.

```
peer-class 50 peer
10.10.10.10/16
10.10.10.10/16
```

Example:
This external network class example includes all other IP addresses.

```
external-class 100 external
0.0.0.0/0
::/0
```

Example:
An example of a policy class is:

```
POLICY policy-class-1
10.10.10.10/1
10.10.10.10/3
10.10.10.11/3
```

1.4.1 Network Class

A network class is a group of subnets that you assign a unique name.

You can use the class names in traffic SandScript, in network protection SandScript, or as reporting dimensions. Sandvine's Network Demographics Server (NDS) can provide reports for network class.

The number of network classes that are created impacts the size of the database that is required for logging, as information is tracked flowing from each network class to every other network class. SandScript fails to reload if you create more than 100 network classes in a subnets.txt file.

Note:
One of the network classes must include the default entries `0.0.0.0/0` and `::/0`. If only one is present, the other is assumed to be in the same network class. If neither is present, the file fails to load.

1.4.2 Policy Classes

A policy class is a group of IP addresses (using subnet notation) that is used to apply differentiated SandScripts.

Using network classes to apply SandScript to a subset of subscribers sometimes has the undesirable side-effect of dividing up subscriber statistics as well. In cases where you do not explicitly want to divide subscriber statistics by class, use a policy class. You can use the policy class to exempt internal mail servers, web servers and, so forth from statistics collection. For example, you can create a policy class called `myportal` to exempt specific traffic from a captive portal.

To create a policy class, in `subnets.txt`, use policy followed by a class name.

Example:
An example of a policy class is:

```
POLICY policy-class-1
10.10.10.10/1
10.10.10.10/3
10.10.10.11/3
```

1.4.3 Border Gateway Protocol

The PTS acts as an Ethernet switch. However, it can use information from the Border Gateway Protocol (BGP) for SandScript decisions and reporting.

SandScript uses BGP data to classify traffic based on its routing information. You can use BGP AS information to execute SandScript based on next-hop AS and other attributes, for example traffic management by AS. For more information see the *PTS SandScript Reference*, section BGP Functions.

This section describes how the PTS uses BGP to dynamically maintain the subnets in `/usr/local/sandvine/etc/subnets.txt`. You can use BGP AS information to populate network or policy classes, along with, or in place of IP addresses. BGP allows the PTS to dynamically update of `subnets.txt`. To use AS numbers in `subnets.txt` and have the PTS listen to BGP updates, you must configure and enable svBGP.

1.4.3.1 Autonomous System Numbers and BGP Communities

If you specify an AS number in `subnets.txt` for a network or policy class, then that class is dynamically updated with the subnets belonging to that AS. You can specify BGP communities in the `subnets.txt` file. If you specify a community, then that class is updated with all subnets for the specified community attribute.

Where there is both a community number and an AS number, the community takes precedence when mapping subnets. If a subnet belongs to two different communities, specifying both communities in different classes of the same class type will cause undefined behavior.

The format for specifying an AS number is `ASX` where `X` is the AS number. For example: AS1000. The format for specifying a community is `ASX:Y` where `X` refers to the AS number to which the community belongs and `Y` refers to the community number. For example: AS1000:300.

BGP community numbers must be in the range 0 — 65535. Be aware that 0, 54272-64511, and 65535 are reserved by the Internet Assigned Numbers Authority (IANA).

Content in subnets.txt	Evaluation	Example
“AS” followed by numbers.	AS numbers.	`AS1000`
"AS" followed by numbers, a colon, and more numbers.	BGP community number.	`AS1000:101`
“AS” followed by a combination of letters and numbers and by the appropriate attribute fields.	Network class name.	`ASWAN 10 internal`
“AS” followed by a combination of letters and numbers but not followed by the attribute fields.	Error.	`ASWAN`

1.4.4 Example subnets.txt Entries

An example of a subnets.txt file is:

Example:

```
#Internal network class with a single subnet.
#Name     Cost Type
internal-1 0 internal
4.1.0.0/24
#Includes a site (session qualifier) which is more specific than the subnet defined below.
4.2.0.0/16; site:2

#Internal network class with multiple subnets.
#Note that the IPs in 4.1.0.0/24 from internal-1 are also in 4.1.0.0/16 below,
#but since 4.1.0.0/24 is a longer prefix, those IPs belong to internal-1.
#Name     Cost Type
internal-2 20 internal
4.1.0.0/16
4.2.0.0/16
4.3.0.0/16
4.4.0.0/16

#Internal network class that uses BGP to automatically discover subnets.
```

```
#Name       Cost Type
internal-bgp 20 internal
AS12345
AS54321:13

#External network class that includes all other IP addresses.
#Since no site is defined, all subscribers in this group
#are treated as though they belong to site 0.
#The file will fail to load without this entry, as it provides a default.
#Name     Cost Type
external 100 external
0.0.0.0/0
::/0

#Policy class
POLICY policy-class-1
6.0.0.0/24
6.0.1.1/32
6.0.1.2/32
```

1.4.5 Editing the subnets.txt File

A default `subnets.txt` file is provided on each PTS element and contains the default configuration. Edit this file to match your network.

Any time a block of IP addresses are added or removed from the network, you must update the `subnets.txt` file by adding or removing those IP addresses.

1. To edit `/usr/local/sandvine/etc/subnets.txt`, in the CLI run: .

   ```
   edit policy subnets
   ```

   The vi editor opens the file for edit.

2. Edit the file with the required changes. When you have completed your edits, exit vi and save your changes with the command:

   ```
   :x
   ```

3. To verify your changes, run the CLI command:

   ```
   reload validate policy <filepath> subnets <filepath>
   ```

4. When you are ready, activate your changes. At the CLI prompt, run:

   ```
   svreload
   ```

5. To refresh the subscriber mappings, run:

   ```
   clear service subscriber-management
   ```

1.5 Domain List File

The domain list file (domains.txt) contains a list of domains and email addresses against which the PTS can perform lookups.

Network protection rules use the domain list file to perform these functions:

- The email spam address matching rule allows for filtering of email addresses under a specific domain or subdomain.
- The DNS domain flood rule examines queries to a specific domain and can ensure that a specific threshold is not exceeded.

Centralized configuration of the domains.txt file is recommended for all deployments. The domains.txt file can be located in a centralized location on an FTP or HTTP/HTTPS server and referenced by all of the elements in the cluster. Centralized configuration ensures that all of the elements in the cluster have access to the same domain list.

While not recommended, the domains.txt file can be maintained on each element in the cluster. The local file is located in `/usr/local/sandvine/etc`.

1.5.1 Syntax of domains.txt

The domains.txt file is composed of a series of single line entries. Each line can contain a single email or domain, a comment, or a group name definition. Group names allow for defining multiple domain lists in a single file. Group definitions are identified by the “group” keyword, followed by a space and the group name. Only alphanumeric characters and these special characters are allowed in email or domain strings and group names: * - _ @ .

Note that:

- Comment lines must start with the # character.
- Empty lines are ignored.
- All input is converted to lower case.
- The * character is only permitted at the front of a domain entry, as a wild card.

This is an example of a domain list which contains acceptable entries and shows the syntax of the entries.

```
group good
domain.com
user@domain.com
*.domain.com
*.domain.name.com
# comments are completely ignored
```

Unacceptable entries have excess spaces, comments on the same line as entries, or illegal characters as illustrated below:

```
corrupt line # comments must be on a separate line
there are spaces in this line
domain.with.illegal&characters
*domain.com
domain.*.com
```

1.5.2 Creating a domains.txt List

To create a domain list:

1. As an administrative user, on any PTS element in the cluster, edit `/usr/local/sandvine/etc/domains.txt`.
2. Add email, domain, and group definitions, one per line.
3. On each element in the cluster, reload the applications by executing `reload`.

1.6 File Validation through Reload

The `reload` CLI command reloads configuration and SandScript policy on the system. Using the `validate` option, you can also use the reload command to validate the syntax of policy.conf files and subnets.txt.

Using the validate option of the reload command does not load or deploy the files. Instead, the tool reports syntax errors and some logical problems. Non-semantic errors, such as using a non-existent field are not identified by the validate option.

Executing `reload` to update configuration parameters automatically uses the validation tool to validate the configuration files before the full reload is attempted. This avoids shunting in the case where there is an obvious problem with the configuration. Any detected errors are included in the `reload` output.

The syntax is:

```
PTS> reload validate policy <filepath> subnets <filepath>
```

Where:

- policy <filepath> is optionally used to specify a file path to the SandScript policy file to validate
- subnets <filepath> is optionally used to specify a file path to the subnets file to validate

By default, the SandScript (`/usr/local/sandvine/etc/policy.conf`) and subnets (`/usr/local/sandvine/etc/subnets.txt`) files are validated. For example:

```
PTS> reload validate
Validating configuration files...
Validating policy...
Validating subnets...
Validation complete, no errors detected
```

To validate files in a non-standard location, use the optional `-policy` and `-subnets` parameters. For example:

```
PTS> reload validate policy /tmp/newPolicy.conf
Validating configuration files...
Validating policy...
Validation complete, no errors detected
```

2

SandScript Grammar

2.1 SandScript Rule Structure

Rules take the format:

```
if <condition(s)> then <action(s)>
```

For example:

```
if expr(Policy.reloaded) then set measurement.SomeMeasurement = 0
```

reads if <condition>, SandScript has been reloaded, is true then perform <action>, reset SomeMeasurement.

General guidelines for rules are:

- Rules are case sensitive. In general, all keywords are lower case with dashes separating words. Always use lower case unless upper case is specified.
- Any line starting with a pound sign (#) is commented out and ignored.
- White space is ignored.
- Rules can be broken into multiple lines by using the backslash (\) to escape the carriage return.
- Strings must be enclosed in single or double quotes. Use single quotes if you want the string to include double quotes and vice versa.
- Any line that does not start with an identifier will give an error in `/var/log/svlog`.

The conventions used when documenting rules are:

- Optional parameters are in braces { }.
- Angle brackets <> are used to identify arguments where you must supply the appropriate value.
- Mutually exclusive options are indicated by the "|" symbol.
- A backslash (\) at the end of a line indicates that the string is continued on the next line.

2.1.1 Rule Evaluation

SandScript policy rules are evaluated on many different types of events, and periodically on existing objects on the element.

A SandScript rule may apply to one or more events, but they are only evaluated on events where they make sense. The set of events where a rule can be evaluated is known as the scope of the rule.

Rules are evaluated at the beginning of each flow once the protocol is recognized, although the SandScript may indicate to continue searching for the right SandScript. All conditions in a rule are tested and when they evaluate to true, the corresponding actions are performed if they are compatible with the actions already performed. If an action is incompatible with actions already performed, then the action is ignored. With respect to incompatible action conflicts, precedence is determined by the order of the rules in the SandScript file.

Traffic rules are re-evaluated throughout the life of a flow. For existing flows, if conditions change, the policy engine applies the appropriate actions such that new actions are applied, and actions whose corresponding conditions are no longer true are retracted. Certain "sticky" actions, such as divert and captive portal, cannot be retracted.

Specifically, traffic rules are re-evaluated when subscriber attribute changes are pushed to the SDE from the SPB.

Specifically, traffic rules are re-evaluated when:

- When a new flow arrives.
- Once the protocol is identified, which may or may not coincide with a new flow event.
- When subscriber attribute changes are pushed to the PTS from the SPB.

- When network protection actions are applied to an IP address. For example, a rule triggers and now traffic needs to be blocked for an IP address.
- When triggered by a previously executed reevaluate action.
- For a background reevaluation.
- At the end of a flow.

Note:
Do not make assumptions about rule evaluation or rely on rules being executed a specific number of times. Reevaluation is a handled by a complex scheduler and is difficult to predict.

Rules that do not have flow conditions or flow actions are optimized into separate contexts that get evaluated on events and objects other than flows. For example, rules that apply to subscribers are periodically evaluated on all active subscriber addresses. Other types of events that cause SandScript to be executed include SandScript reload, the receipt of a diameter message, or the expiration of a SandScript timer.

2.1.2 Logic

Modifiers can be used to specify logic operations in rules.

The available operators are:

- `and` - evaluates to true if both conditions are met
- `or` - evaluates to true if either condition is met
- `not` - evaluates to true if the condition is not met

When using more than one logical operator, use parentheses () to indicate precedence. The `and` operator takes precedence over the `or` operator.

Example Conditions

not condition

```
if not protocol "http" then divert
```

condition1 and condition2

```
 if server "servicelevel" = "bronze" and protocol "http" \
then shape to client shaper "httpserver" unique by Flow.Client.Stream.Http.Host
```

condition1 or condition2

```
 if protocol "bittorrent" or protocol "gnutella" \
then divert
```

(condition)

```
if not ((protocol "bittorrent" or protocol "gnutella") and \
(client "servicelevel" = "bronze" or server \ "
servicelevel" = "bronze")) then count
```

Actions

Rules can cause multiple actions to take place. To accomplish this, an `and` operator can be used with compatible actions.

2.1.3 SandScript Policy Groups

Rules can be organized into policy groups.

A policy group is a method of grouping a number of rules where the rule conditions are evaluated and by default only the actions of one rule in the list are performed. The actions specified in the first rule that matches the condition are performed and any remaining rules in the list are skipped. This speeds up processing. For example, a policy group could be used to apply rules based on time of day criteria.

Policy group with conditions

Conditions can be placed directly on policy groups. The syntax is:

```
PolicyGroup condition(s) {
    rules and/or nested policy groups...
}
```

For example:

```
PolicyGroup all {
    integer "var1"
    string "var2" = "Value"
    boolean "var3" = expression
    shared integer "sharedVar" = 180
    if condition(s) then set var1 = expression
    if condition(s) then set expression = var2
    if expr(var3) then action(s)
}
table "MyTable" (integer "key", integer "value") timeout none \
    unique by (Table.MyTable.Key)
PolicyGroup all {
    Table.MyTable:cursor "row1"
    Table.MyTable:cursor "row2" = Table.MyTable[2]
    Table.MyTable:cursor "row3" = Table.MyTable.Create(2)
    if condition(s) then row2.Set(value: 7)
    if condition(s) then set row3.value = 180
    if condition(s) then set row1 = row2
    if expr(row2 = row3) then action(s)
    if expr(row1.value > 10) then actions(s)
}
```

The rules within the policy group are only evaluated if the group condition is matched. By default, a policy group stops evaluating its rules when the first match is found, essentially creating an if/elseif/else... decision tree. When a matching rule is found, its actions are performed and any remaining rules and groups are skipped. If the condition of a nested policy group evaluates to true, this also causes any remaining rules and groups to be skipped.

The scope of a policy group is the intersection of the scope of all the rules/groups inside. So a single scope-restricting field nested deep in the group forces the whole group to have the scope of that field. If the intersection results in the empty set, there is a "conflicting scopes" error on reload.

Example:
In this example, if the client's service level is something other than gold, no actions are performed for HTTP flows because the condition of the inner policy group causes the remaining rules in the outer group to be skipped (even though the inner rule may evaluate to false).

```
PolicyGroup {
    PolicyGroup protocol "http" {
        if client "servicelevel"="gold" then action1
    }
    if true then action2
}
```

Example:

In this example, the policy group ensures that the action applies to the user Bob, and only Bob. For a login event, action 1 is applied, for a logout event, action 2 is applied, if it is neither of these then action 3 is applied and, if the user is not Bob, action 4 is applied.

```
PolicyGroup {
    PolicyGroup expr(RADIUS.'Accounting-Request'.'User-Name' = "Bob") {
       if expr(RADIUS.'Accounting-Request'.'Acct-Status-Type' = 1) then action1
       if expr(RADIUS.'Accounting-Request'.'Acct-Status-Type' = 2) then action2
       if true then action3
     }
    if true then action4
}
```

“all” policy group

An "all" policy group evaluates all the rules in the group if the group condition matches. This is a convenient way to write multiple rules that share one or more common conditions.

```
PolicyGroup all <condition(s)> {
    rules and/or nested policy groups...
}
```

Example:
This example session manages specific traffic. Without the `all` option, the rule and policy groups in this example are evaluated in order, provided that the time conditions (weekdays from 7 am to 4 pm) listed in the parent policy are both met. Since there is no rule that has true as a condition or policy group that is not conditioned, rules/groups are read in order and exit on the first match.

If the flow protocol is BitTorrent or eDonkey, and the direction is unknown, the policy group exits on the first rule since it matches. When the direction becomes known, the first condition is not met. The policy groups are vetted in order and as they are mutually exclusive, either one of the conditions are met, or the flow falls through having met no conditions on either rules or policy groups.

```
PolicyGroup time wday mon-fri and time hours 0700-1600 {
    if expr(Flow.IsApplicationProtocol("bittorrent","edonkey") and transfer unknown \
    then continue
    PolicyGroup expr(Flow.ApplicationProtocol = Protocol.eDonkey) {
     if transfer uni and \
  (client class "internal1" or server class "internal1") \
  then tcp_reset
 }
 PolicyGroup expr(Flow.ApplicationProtocol = Protocol.BitTorrent) {
  if transfer uni and \
  (client class "internal1" or server class "internal1") \
  then tcp_reset
 }
}
```

Example:
This example uses the all option. While the net effect is the same as the previous example, the latency is longer in this case if the flow direction is unknown. The containing policy group does not exit but all the conditions on all nested policy groups are evaluated.

```
PolicyGroup time wday mon-fri and time hours 0700 - 1600 all {
 if expr(Flow.IsApplicationProtocol("bittorrent","edonkey") and transfer unknown \
  then continue
 PolicyGroup expr(Flow.ApplicationProtocol = Protocol.eDonkey) {
      if transfer uni and \
   (client class "internal1" or server class "internal1") \
   then tcp_reset
 }
 PolicyGroup expr(Flow.ApplicationProtocol = Protocol.BitTorrent) {
  if transfer uni and \
   (client class "internal1" or server class "internal1") \
   then tcp_reset
```

```
    }
  }
```

2.1.4 Include Statements

SandScript policy files can be nested.

By adding an include statement to the SandScript file, additional SandScript files can be implemented. The file at `/usr/local/sandvine/etc/policy.conf` is always loaded. Other SandScript files are only loaded if they are included in the `/usr/local/sandvine/etc/policy.conf` file or nested within files included in it.

Include statements have the syntax:

```
include "<filepath>"
```

Where <filepath> is the full path name of the file to include. For example, "/usr/local/sandvine/etc/policy2.conf"

Include statements can appear anywhere in a SandScript policy file. The rules contained in the included file are evaluated in order, starting from the location where the file is included.

Including a file this way effectively inserts the SandScript file content at that location in the original SandScript file. Often one of the motivating factors for using multiple SandScript files is to hide complicated application details from a simpler, presentable SandScript file with declarations, or constants, that let you configure the application. In early product release this was typically a list of macros/defines. Unfortunately, extensive use of most macros often causes degraded SandScript performance. In particular, macros are often complex expressions that refer to other macros—sometimes many levels deep. In most of these cases local variables that build on one another perform much better.

Since you have to define local variables inside a SandScript policy group, you can include SandScript files within the context of a SandScript policy group. This means the included file's content is inserted at that point in the SandScript search tree, and can make use of local variables in that scope. It can also define new local variables that the original file can use (while still in scope).

2.2 Attributes

Attributes provide a mechanism for associating data with a subscriber or with a subscriber sessions. SandScript can be used to define attributes and to associate them with one or more subscribers.

The attributes that are attached to subscribers can be used to trigger SandScript-defined management of the subscribers and their flows. The limitations for attributes are:

- An attribute name cannot exceed 128 characters.
- An attribute value cannot exceed 4000 characters.
- Attribute names must start with a letter or an underscore and can only contain alphanumeric characters and underscores (_).
- The maximum number of attributes that may be defined in SandScript is 1000.

Attributes have a few important properties which are described below.

Enumerated versus Non-enumerated

Enumerated attributes are string-based attributes that can only take on certain pre-defined values. These values are defined in SandScript when the attribute is declared.

Non-enumerated attributes are typed attributes that can assume any value that is valid for their type. Supported types are strings, integers, floating point values, timestamps (ISO-8601), IP addresses, and booleans.

Local versus Non-local

Local attributes are lightweight attributes that are not persisted to the subscriber database. This causes them to consume fewer system resources. However, the lack of persistence means that local attributes cannot be reported on using Network Demographics, and these attributes are only defined on the element on which they are created (not cluster-wide).

Non-local attributes are persisted to the subscriber database, making them more resource-intensive. However, they can be reported on and they are cluster wide.

Subscriber-based versus Session-based:

Subscriber attributes are associated with a particular subscriber and all of the subscriber flows for the lifetime of the subscriber.

In contrast, a session attribute is associated with a single subscriber session. When that session ends, the attribute is no longer defined, even if the subscriber has other active sessions. Subscriber attributes are writable in SandScript but session attributes are read-only in SandScript. Session attributes are typically gleaned from a DIAMETER/RADIUS login session.

Note:
Session attributes are only supported when all SPBs in the cluster node are of release 5.60 or greater.

Record Attributes

Record attributes are a collection of attributes under one name. Each attribute in the collection can have its own type and store a unique value. These attributes correspond to RADIUS-mapped or DHCP-mapped attributes on the SPB, where multiple values from the RADIUS or DHCP packet are stored in a single SPB attribute.

Record attributes are read-only and cannot be defined as local attributes. Additionally, record attributes are only accessible through SandScript fields - see the PTS or SDE *SandScript Reference* for more information on the `Subscriber.Attribute` and `Session.Attribute` fields.

2.2.1 Declaring Attributes in SandScript

Attribute declarations must precede their use elsewhere in SandScript policy.

2.2.2 Enumerated Attributes

```
attribute {subscriber|session} "<attribute_name>" values "<value1>" "<value2>" "<value3>" …
```

Where:

- {subscriber|session} defines whether the attribute is attached to a subscriber or a session. Defaults to subscriber.
- <attribute_name> is the name of the attribute
- <value1> and so on are the values.

For example:

```
# Declare a subscriber attribute that describes a subscriber's bandwidth usage.
# This is a subscriber attribute because the session keyword was omitted.
attribute "subscriber_bandwidth" values "light_user" "heavy_user" "abuser" "top_talker"
# Declare a session attribute for Diameter Credit Control Request (CCR) types
attribute session "diameter_ccr" values "initial" "update" "termination"
```

2.2.3 Non-enumerated Attributes

```
attribute {subscriber|session} "<attribute_name>" type <attribute_type>
attribute {subscriber|session} "<attribute_name>" type record \
 (<attribute_type> "<field_name_1>" {, <attribute_type> "<field_name_2>" ...})
```

Where:

- {subscriber|session} defaults to subscriber.
- <attribute_name> is the name of the attribute.
- <attribute_type> defines the type of the attribute. Supported types include string, integer, float, boolean, timestamp, ipaddress.

For example:

```
# Declare an integer attribute to count a subscriber's flows
attribute "subscriber_flow_count" type integer
# Declare a timestamp attribute to store a subscriber's login time
attribute session "login_time" type timestamp
# Declare a record attribute to store radius-mapped attributes for a subscriber session
attribute session "radius_info" type record ( \
    ipaddress "framed_ip_address", \
    string "account_session_id", \
    integer "calling_station_id" )
```

Type ranges for non-enumerated attributes are:

- Integer values can range from -9223372036854775808 to 9223372036854775807.
- Float values can be defined with values such as 3.1415926 or 1e10.
- Boolean attributes are stored in database as either true or false. These values are interpreted as true:
  - "t" or "T"
  - "true" or "TRUE"
  - "y" or "Y"
  - "yes" or "YES"
  - any non-zero number, such as "1", "1000", "+1", "-1", or ".5"
- IP Addresses support IPv4 and IPv6 addresses. IPv6 addresses with the IPv4-mapped IPv6 prefix of 0:0:0:0:0:ffff::/96 are interpreted as IPv4 addresses. IP address type cannot be converted to or from any other field type except string.

2.2.4 Timestamp Attribute

Timestamp is a data type. Timestamp attributes are integers which represent the number of seconds since the UNIX epoch (January 1, 1970 at 00:00:00 UTC).

```
attribute "<attribute_name>" type timestamp
local_attribute "<attribute_name>" type timestamp
```

Where:

- Timestamp attribute values are stored in the database in ISO-8601 compliant format yyyymmddThhmmssZ always as UTC. For example, 20071231T2359Z. Timestamp attributes are equivalent integer attributes.
- Timestamp string parsing supports a subset of ISO 8601, date followed by time followed by time zone specification.
- Only the date part is required. Everything else is optional.
- Dates may be expressed as either yyyy-mm-dd or yyyymmdd.
- The time component is optional. If present, it follows the date. The date and time are separated with the letter T.

- Minutes and seconds are optional. Hour, minute, and seconds components must have exactly 2 digits.
- Time specification may contain fractional seconds after a dot. This is ignored when parsing (for example, 23:59:59.999).
- If the time zone is not included, UTC is assumed.
- Two formats are supported. The letter Z which means UTC, or the offset in hours and minutes from GMT. The colon (:) is optional.

Example:
UTC time examples:

```
20071231T235959Z means UTC
20071231T235959 means UTC (by default)
20071231T235959+02:00 means UTC + 2 hours
20071231T235959-0430 means UTC - 4 hours and 30 minutes
20071231T235959Z-06 means UTC - 6 hours
```

Local time examples:

```
2007-12-31T23 means Dec 31, 2007 23:00:00
20071231T23:59 means Dec 31, 2007 23:59:00
20071231T2359 means Dec 31, 2007 23:59:00
20071231T23:59:59 means Dec 31, 2007 23:59:59
20071231T235959 means Dec 31, 2007 23:59:59
```

2.2.5 Local Attributes

Local attributes have similar syntax to non-local attributes, except they begin with the local_attribute keyword. In addition, local attributes are always session-based since they are not intended to be persistent.

```
local_attribute {session} "<attribute_name>" values "<v1>" "<v2>" "<v3>" "<v4>" …
local_attribute {session} "<attribute_name>" type <attribute_type>
```

Where <attribute_type> defines the type of the attribute. Supported types include string, integer, float, boolean, timestamp, ipaddress.

Example:

```
local_attribute "subscriber_type" values "light_user" "heavy_user" "abuser" "top_talker"
local_attribute "local_login_time" type timestamp
```

2.2.6 Session Attributes

The optional {session} modifier can be used in an attribute declaration to specify if the attribute is session-based or subscriber-based.

The primary difference between these two modes is the lifetime of the attribute. Subscriber-based attributes can be accessed in SandScript as long as a subscriber is mapped. However, session-based attributes are defined only for a single subscriber-ip session. Note that local attributes are always session-based, but all other attribute types can be either session or subscriber based.

Example:

```
attribute session "subscriber_bandwidth" values "light_user" "heavy_user" "abuser"
attribute subscriber "subscriber_flow_count" type integer
local_attribute session "local_login_time" type timestamp
```

2.3 Subscriber Classes

The subscriber_classes statement specifies which network classes or policy classes (from subnets.txt) contain subscriber IP addresses.

Only addresses from these classes are looked up in the subscriber-IP map.

Note:
subscriber_classes must be declared prior to any subscriber scoped policy.

This example indicates that the policy engine should look up IP addresses in the "internal1" and "internal2" network classes in the IP-subscriber database.

```
# cost-classes to be looked up in IP-subscriber database
subscriber_classes "internal1" "internal2"
```

2.4 Node Qualifiers

Node qualifiers are used to apply conditions, such as class or port, to a particular node of a flow.

A node is the endpoint of a flow. Each flow has two nodes - for this reason, node qualifiers come in pairs. Certain conditions limit the node qualifiers allowed - see each condition for a description of which node qualifiers are permitted. Valid node qualifiers are:

subscriber and internet
client and server
sender and receiver
source and destination

The node qualifier designation is determined by protocol handlers. Only these protocols should be used for sender and/or receiver:

Ares
BitTorrent
eDonkey
Gnutella

2.4.1 Subscriber/internet

Sandvine recommends using the subscriber and internet node qualifiers.

- Subscriber - Refers to the endpoint of the flow attached to a subscriber interface of the PTS.
- Internet - Refers to the endpoint of the flow attached to an internet interface of the PTS.

2.4.2 Client/server

Every flow has one client and one server.

- Client - The initiator of the flow (the node that sends the first packet, for TCP the sender of the SYN).
- Server - Opposite of client, the acceptor of the flow (receives the first packet, for TCP the receiver of the SYN, sender of the SYN-ACK).

2.4.3 Sender/receiver

The sender and receiver node qualifiers are related to the data transfer direction of the flow. For more information about data transfer direction see *Transfer* on page 38.

- Sender - Refers to the host or hosts that are transmitting data. Unidirectional flows have one sender. Bidirectional flows have two senders (both hosts are senders since data is being transmitted in both directions). Flows with no direction (no bulk data) have no sender.
- Receiver - Opposite of sender, refers to the host or hosts that are receiving data. Unidirectional flows have one receiver. Bidirectional flows have two receivers (both hosts are receivers since data is being received in both directions). Flows with no direction (no bulk data) have no receiver.

The relationship between client/server, and data transfer direction sender/receiver is:

Transfer Direction					
Host		uni to client	uni to server	bidirectional	no direction
	client	receiver	sender	sender and receiver	
	server	sender	receiver	sender and receiver	

2.4.4 Source/destination

The source and destination node qualifiers can only be used when defining a bitrate/byte/packet measurement or bitrate limiter.

- Source - Indicates that the bitrate to be measured/limited is transmitted by the indicated port/class.
- Destination - Is the opposite of source, and it indicates that the bitrate to be measured/limited is received by the indicated port/class.

2.5 Conditions

Conditions are used to identify traffic or subscribers to which specific actions should be applied.

Conditions can be combined or modified using logical operators. Brackets () should always be used to indicate precedence.

2.5.1 Any

A token that can be used in a condition.

It evaluates to true if the expression has a non-NULL value. It should not be used in quotes. For example:

```
if time wday = any
```

2.5.2 Attribute Condition

The attribute condition matches a subscriber attribute.

```
sender|receiver|client|server|subscriber "<name>" = "<value>"
```

You may see this condition in legacy SandScript. SandScript now uses the expression syntax for subscriber attribute matching. For more information see *Expressions* on page 35.

Example:
An example of using an expression is:

```
if expr(Subscriber.Attribute.<name> = <value>) then <action>
```

2.5.3 Class

The class condition is the name of a network class or policy class as defined in the subnets.txt file for the cluster.

```
sender|receiver|client|server|subscriber|internet class "<classname>"
```

Where:

- sender|receiver|client|server|subscriber|internet is the node qualifier for which node to test
- <classname> is the name of the network class or policy class as defined in the subnets.txt file.

Example:
This example limits traffic to peers differently than to external network classes.

```
#Policy Group 1
PolicyGroup{
 if receiver class "peer" then allow
 if receiver class "external" then tcp_reset
}
```

2.5.4 Expressions

Expressions are extended SandScript policy syntax for accessing the fields provided by SandScript.

A SandScript expression takes the form of a condition where it is enclosed within an `expr( )` operator. Expressions are defined in terms of fields, constants, operators, and other expressions. For detailed information on expressions, refer to the SDE or PTS *SandScript Reference*.

Example:
This expression marks flows that are older than 2 minutes (120 seconds) per dscp 27.

```
if expr(Flow.Age>120) then mark dscp 27
```

2.5.5 IP Address

The ip_addr condition is used to target a rule specifically at an IP address.

```
sender|receiver|client|server|subscriber|internet ip_addr <ipaddress>
```

Where:

- {sender|receiver|client|server|subscriber|internet} is the node qualifier for the IP address
- <ipaddress> identifies the IP address at which the rule is targeted

Example:

```
if client ip_addr 192.168.10.2 then captive_portal \
 "http://myportal.domain.com/expired_account.rvt"
if client ip_addr 192.168.2.4/24 then tcp_reset
```

2.5.6 Layer 4 Protocol

The layer4protocol condition is used to target a rule specifically at layer4 protocols such as TCP. This is from the protocol field in the IP header.

```
layer4protocol <protocol>
```

Where <protocol> is one of TCP, UDP, ICMP, GRE, AH, ESP, SCTP, or IP_IN_IP or an IANA protocol number (for example, SCTP =132).

Example:
For example:

```
if layer4protocol TCP then mark dscp 27
```

2.5.7 Port

The port is the physical port on the element, which is also called the interface.

```
sender|receiver|client|server port subscriber|internet
```

For PTS 8210 only:

```
sender|receiver|client|server|subscriber|internet port <number>
```

Where:

- sender|receiver|client|server|subscriber|internet is the node qualifier
- internet|subscriber refer to the internet and subscriber facing ports
- <number> refers to a physical port on the element for PTS 8210 only.

Example:
The following example applies shaping to subscribers (on port 3) running HTTP servers. The rule matches when the HTTP server is on port 3 and an HTTP GET is identified. The action is to apply shaping in the direction of the client and allocate a unique shaper per subscriber HTTP server.

"server port 3" means that the packet received on port 3 has the server's IP address as the source. The packet sent on port 3 would have the server's IP address as the destination.

```
if server port 3 and protocol "http_get" then shape to \
 client shaper "httpserver" unique by Flow.Client.Stream.Http.Host
```

2.5.8 Protocol

Protocol is the application protocol and is identified by strings such as "bittorrent" or "http".

```
protocol "<name>"
```

Where <name> is a protocol string. For a list of supported protocols and associated strings, refer to the Loadable Traffic Identification Package.

Each supported protocol and sub-protocol has a string that is used to identify the protocol in policies. For example, these strings are used to identify the eDonkey protocol:

Protocol	Sub-protocol	SandScript String
eDonkey	---	edonkey
	eDonkey Control	edonkey_control
	eDonkey Data	edonkey_data
	eDonkey UDP	edonkey_udp
	eDonkey Encrypted Control	edonkey_encrypted_control
	eDonkey Encrypted Data	edonkey_encrypted_data
	eDonkey Encrypted UDP	edonkey_encrypted_UDP

Example:
In this example, subscriber HTTP clients who are identified as abusers are directed to a captive portal.

```
if protocol "http_get" and client "abuser"="true" then \
 captive_portal "http://www.myISP/mynotice/"
```

2.5.9 Provider

The provider condition pertains to VoIP protocols only and can be used to identify a specific VoIP provider.

```
provider "<provider_name>"
```

Where <provider_name> is a VoIP provider. For information on identifying VoIP providers, see *Configuring VoIP Measurements and Management* on page 244

Example:
In this example, if the provider is “Myprovider”, the flow is session marked.

```
if provider "Myprovider" then mark dscp 27
```

2.5.10 Time of Day

The time of day condition lets you limit an action to a specific time frame.

```
time wday <wday_list>
time wday <wday_list> hours <hhmm-hhmm> {hours <hhmm-hhmm>}…
time hours <hhmm-hhmm> {hours <hhmm-hhmm>}…
```

Where:

<wday_list> Days of the week are specified using a three character abbreviation for the day (Mon) or by spelling out the entire day (Monday). The first character in the string can be either upper or lower case, with the remaining characters being lower case (Mon is acceptable, MON is not).

Week day abbreviations are Mon, Tue, Wed, Thu, Fri, Sat and Sun. A range of days is specified using a dash (Mon-Fri, Friday-Monday). For example:

- wday Mon, Wed, Fri indicates Monday, Wednesday and Friday
- wday Wednesday-Saturday indicates the range from Wednesday to Saturday

<hhmm-hhmm> {GMT} Hours when the condition is valid is indicated by a range. Time conditions may be followed by GMT which indicates that days and times are in Coordinated Universal Time (UTC). When GMT is not present, local time is assumed. Note that only local time and UTC are supported.

For example: hours 0800-1800 indicates the hours from 8 AM to 6PM, local time. 2400 can not be used to indicate midnight. For example, to specify 5 PM to midnight, use 1700-0000.

Note that if a weekday and hours are both specified, the hours can extend beyond the specified week day. For example if Friday and a time of hours 1800-0200 is specified, the rule will extend from 6 PM on Friday to 2 AM on Saturday.

Example:
In this example, subscribers in the bronze class are shaped during prime time hours - Monday to Friday from 1800 to 0200 and on weekends.

```
if sender "servicelevel"="bronze" and (time hours 1800-0200 \
 or time wdays sat, sun) then shape to client shaper \
 "httpserver" unique by Flow.Client.Stream.Http.Host
```

2.5.11 TCP Port

The tcp_port condition lets you limit an action to a range of TCP ports.

```
sender|receiver|client|server|subscriber|internet tcp_port port|port_range
```

Where:

- sender|receiver|client|server|subscriber|internet is the node qualifier for the node to test
- port|port_range is the port or port range to evaluate. A range is indicated with a hyphen (-) between values, no spaces. For example: 80-90.

Example:

```
if server tcp_port 80 then divert destination "cache"
```

2.5.12 Transfer

The transfer condition is used to select flows with a particular data transfer direction.

```
transfer unidirectional|bidirectional|none|unknown
```

Where:

- unidirectional refers to flows where data is being transferred in a single direction, for example a Gnutella upload or download
- bidirectional refers to flows where there is data being transferred in both directions, as is commonly the case for peer-to-peer protocols like BitTorrent and eDonkey
- none refers to flows that are not transferring a significant amount of data in either direction, as is the case for peer-to-peer control flows - typically flows with no direction are excluded from session management policies
- unknown refers to flows for which the direction has yet to be determined. Some protocols must be inspected deep into the flow before the direction is known. This is typically only used to control the continue action which instructs the PTS to continue to inspect the flow until the direction is known.

The keywords **unidirectional** and **bidirectional** can be abbreviated to uni and bi.

The transfer direction for a flow can change during its lifetime, possibly multiple times. The direction is initially based on recognition (the analysis of the binary data) and then is updated by monitoring the amount of data that is transferred in each direction. For example, recognition may determine that a flow is bidirectional to begin with, but if the data being transferred in one direction stops, the flow will be reclassified as unidirectional.

Example:
This example assumes a limiter that is defined elsewhere:

```
PolicyGroup protocol "edonkey" {
    # keep inspecting the flow until the direction is known
    if transfer unknown then continue
    # session managing all unidirectional flows being sent to an external address
    if transfer uni and receiver class "external" then tcp_reset
    # allow up to 100 bidirectional flows
    if transfer bi and expr( limter.edonkey_bi.limit ) then tcp_reset
}
```

2.5.13 True

The true condition indicates that the rule is always true.

```
if true then <action>
```

2.5.14 UDP Port

The udp_port condition lets you limit an action to a range of UDP ports.

```
sender|receiver|client|server|subscriber|internet udp_port port|port_range
```

Where:

- sender|receiver|client|server|subscriber|internet is a node qualifier for the node to test
- port|port_range is the port or port range to evaluate. A range is indicated with a hyphen (-) between values, no spaces. For example: 80-90.

Example:

```
if client udp_port 1234 then mark dscp 25
if server udp_port 12-20 then mark dscp 29
```

2.6 Actions

All SandScript rules must have an action. Use actions to specify a behavior when a condition is matched. Multiple actions can be associated with a single condition by combining them using the `and` operator.

```
if <conditions> then <action> {and <action>}
```

2.6.1 Limitations for Multicast and Broadcast Packets

L2 (Ethernet) multicast and broadcast packets cannot have actions applied to them. Regardless of the condition, the Sandvine system cannot perform any action as these packets are treated in a special way, and are not inspected.

2.6.2 Allow

The allow action permits the flow to be bridged, unhindered through the PTS.

Allow prevents any subsequent divert or block.

Example:

```
if sender "subscriberclass"="platinum" then allow
```

2.6.3 Analyze

The analyze action is used to trigger deeper analysis of protocols for which there is support in the PTS.

For protocols that support analysis, fields are made available in SandScript for the remainder of the flow until the analyze action is no longer encountered. The analyze action has no effect on flows that started before a PTSD restart. As a result, PAL fields that depend on the analyze action are not defined for those flows.

The supported protocols are:

Protocol	AnalyzerName	Enabled with “if true then analyze...”
HTTP	Http	yes
DNS	Dpm	no
RTSP	Rtsp	yes
RTMP	Rtmp	yes
Video	Video	no

Example:
This SandScript enables analyze for the protocols indicated in the table above.

```
if <condition> then analyze
```

Optionally, you can specify the protocol to analyze using this syntax:

```
if <condition> then analyze analyzer <"analyzer-name">
```

In these SandScript examples, the protocol of the flow is used to potentially map to the analyzer registered for that protocol.

```
if expr(Flow.SessionProtocol = Protocol.Http) then analyze analyzer "Http"
if expr(Flow.SessionProtocol = Protocol.Rtsp) then analyze analyzer "Rtsp"
```

For additional examples of how to use the analyze action, refer to the http_response action (see *HTTP Response* on page 49).

Note:
You can set `Flow.Stream.ReadOnly = false` if you want to apply actions, such as shape and block, faster than if these actions were based on stream analyzer conditions. A flow starts as read-only in its first policy run, but if at any point the `Flow.Stream.ReadOnly` is not set to 'false', the flow will become read-only forever. When the flow is not read-only, the PTS buffers packets to reassemble the stream to analyze HTTP. If a flow continues in non

read-only mode, latency is added to the HTTP connection. This is a problem in case of lost or reordered packets. Set the `Flow.Stream.ReadOnly` value to "true" in the SandScript as soon as possible.

2.6.4 Block

The block action drops packets of the flow.

```
block {permanent false|true}
```

Where permanent is an optional parameter that defaults to false. If true, the block action cannot be removed from the flow and all packets in the flow are dropped. If false, the block action is removed if subsequent SandScript evaluation does not issue the block action.

Example:
For example, to block all Gnutella uploads:

```
if protocol "gnutella" and sender class "internal" then block
```

2.6.5 Captive Portal

The captive_portal action redirects the user to a specified URL. How frequently this occurs is part of the action.

```
captive_portal "<name>" {cycle <cycle_time>} {delay <delay_time>} {interferences <number>}
captive_portal full_response "<string>" {cycle <cycle_time>} {delay <delay_time>} {interferences
 <number>}
send_response "<string>" {cycle <cycle_time>} {delay <delay_time>}{interferences <number>}
```

<name>

The URL (http://...) to redirect the user to.

Note:
There is a limit of 1024 characters in redirected URLs.

cycle <cycle_time>

Indicates how long to track state. The state is deleted when cycle time expires and starts all over if there is more traffic from the user. This can be set to seconds, hours, or minutes.

60seconds	60sec
60minutes	60min
1hour	1hr

delay <delay_time>

Indicates how long the PTS waits before interfering. This is the number of seconds or minutes of inactivity that must elapse before the user is redirected. This prevents the HTTP GET from interfering with other activities. This can be set to seconds, hours, or minutes.

60seconds	60sec
60minutes	60min
1hour	1hr

interferences <number>

Indicates how many times should the PTS get in the way during that cycle. This is the number of redirections in a row to the captive portal per the cycle interval.

full_response <string>

Specifies a string that is sent in response to the captive portal packet.

send_response <string>

Injects any static string into a flow. Note that expressions cannot be used to build the response and that using send response has the same effect as using captive_portal full_response.

Note that:

- If <cycle_time> and <delay_time> are set to 0, the user is redirected to the captive portal each time they attempt an HTTP GET.
- Using ${SUB} in the URL will cause the name of the client subscriber (or empty if not available) to be placed at this point in the redirected URL.
- Using ${IP} in the URL will cause the IP address of the client to be placed in the redirected URL.
- Using ${<attr_type} will cause the designated attribute (for example, ${tier} if there is an attribute called ‘tier’) to be placed in the redirected URL. Note that only one attribute can be used in conjunction with captive portal. If the attribute string is not available, it will be empty.

Example:
This example causes the PTS to interfere with every single session that users in the category make.

```
if subscriber "abusers" = "true" and protocol "http_get" \
  then captive_portal "http://server.../sid=${SUB}"
```

Example:
This example results in partial interference: twice an hour spaced at least 5 minutes of inactivity apart. This type of action is useful if the user is at 90% of a limit and you want to inform them of pending action.

```
if subscriber "abusers" = "true" and protocol "http_get" \
 then captive_portal "http://myserver.com/sid=${SUB}" \
 cycle 60min interferences 2 delay 5min
```

Example:
In this example, if the client sends an HTTP GET request the PTS responds with the HTTP 200 OK response. The send_response action and the captive_portal action with the full_response option result in the exact same behavior in the PTS. Both also work inline and offline. In offline mode, packets are sent to the client and server via a next hop router through the service plane.

```
# this will respond to the client with a 200 OK
if protocol “http_get” \
then send_response “HTTP/1.1 200   OK\r\n\r\n”
```

Example:
This example will captive portal HTTP GET traffic. For information on using define, see the *PTS SandScript Reference*.

```
# Define Objects to use
define "is_http_get" = \
  (Flow.SessionProtocol = Protocol.Http and
    Flow.Client.Stream.Http.Command = "GET")
if expr(is_http_get) then captive_portal "http://x.x.x.x/portal"
```

Example:
This example will captive portal all HTTP traffic and not just the HTTP GETs.

```
#Define Objects to use
define "is_http" = \
 (Flow.SessionProtocol = Protocol.Http)
if expr(is_http) then captive_portal "http://x.x.x.x/portal"
```

2.6.6 Continue

The continue action indicates that the flow will continue to be inspected until the continue action is no longer issued. Normally, inspections stop as soon as the protocol is identified.

Example:

```
if protocol "bittorrent" and transfer unknown then continue
```

2.6.7 Count

The count action causes statistics to be written to the database.

```
count
count demographic
count [client|server|sender|receiver] subscriber basic|protocol
count voip_qoe
```

To collect port-based statistics for total traffic or for protocol flow by port:

```
count
```

To collect statistics for network class-to-network traffic as well as basic port statistics:

```
count demographic
```

To collect statistics relating to a particular node, protocol, and/or subscriber in addition to port statistic, the syntax is:

```
count client|server|sender|receiver subscriber basic|protocol
```

Where:

- client|server|sender|receiver is the node qualifier.
- basic|protocol indicates how to collect subscriber statistics.
- If count subscriber is specified, subscriber_classes must be defined to identify the network classes in which the subscribers reside.
- If count subscriber basic is specified, by default the PTS only exports the data once every hour. You can configure the default statistics logging interval (in seconds) for subscriber basic statistics using the `set config service statistics log-interval subscriber` CLI configuration command.

To enable the VoIP feature:

```
count voip_qoe
```

This action analyses VoIP calls and collects a variety of information relevant to the call, as well as a variety of quality of experience (QoE) results. Note that:

- A VoIP QoE license is required for this feature.
- Subscribers of interest must be mapped.
- This feature only works in conjunction with SPB 2.01 or higher.
- This action is enabled on a protocol by protocol basis.

Example:
This example collects detailed subscriber statistics for subscribers that have the “substats” attribute set to “detail” and the basic subscriber statistics for everybody else. Note that the subscriber_class declaration must occur prior to the use of any subscriber scope SandScript.

```
subscriber_classes "userclass1" "userclass2"
attribute "substats" values "detail" "basic"

PolicyGroup
{
 if subscriber "substats"="detail" then \
  count subscriber protocol
 if true then count subscriber basic
}
```

Example:

These examples enable the VoIP QoE feature for specific protocols. Including the first line in the SandScript file enables the VoIP QoE feature for SIP traffic; including the second line in the SandScript file enables the VoIP QoE feature for MGCP traffic.

```
if protocol "sip" then count voip_qoe
if protocol "mgcp" then count voip_qoe
```

2.6.8 Decrement

The decrement action decrements the value of the specified SandScript field.

Decrement is only compatible with writeable integer fields. By default, the value of the field is decremented by 1. The optional `by` syntax can be used to decrement the value by some other value. When it is used to decrement the value of a subscriber attribute, the attribute expiry time can be set using the `for` syntax.

```
decrement <field> {by <expression>} {for <time>}
```

Where:

field	An integer SandScript field that can be set
expression	An integer SandScript expression that defines the value to decrement by.
time	An integer SandScript expression plus a unit of time, for example, “10 minutes” or “LocalTime(2010,01,01) - Now seconds”. Specifies when a subscriber attribute decremented by the action will expire and become null.

2.6.9 Delete Row

Use the delete_row action to explicitly delete rows from a table.

```
delete_row <row1> {, <row2>} {, <row3>} ...
```

Where the rows are SandScript fields representing table rows. If the table is implicitly indexed, you do not have to specify the row. For example:

```
Table.<TableName>
Table.<TableName>{Key}
cursorLocalVariable
```

2.6.10 Deserialize

Use the deserialize action to deserialize binary data into writable SandScript fields according to a specified format. This action is the logical inverse of the serialize function.

```
deserialize( \
 input:input_expression, \
 format:format_expression, \
 'output{0}':output_expression, \
 'output{1}':output_expression, \
 'output{2}':output_expression, ...)
```

Where:

input_expression	The expression to deserialize.
format_expression	The format of the input.
output_expression	An expression used to store the output.

Each deserialized value is assigned to an output expression. Output expressions must be writeable (for example, local variables) and compatible with the types specified in the format string. The number of output expressions must exactly match the number of items to be deserialized as specified in the format argument.

When deserializing an input that does not have enough data to fill all output expressions, as many expressions as possible will be deserialized and the rest will be set to null.

2.6.10.1 Format String

Each format character in a format string, except padding, requires an additional output expression to be provided as an argument to the deserialize action. The format string must be composed of these format characters:

Type	Characters	Notes
Integer	• c - 8 bit integer • s - 16 bit integer, little endian • S - 16 bit integer, big endian • i - 32 bit integer, little endian • I - 32 bit integer, big endian • w - 64 bit integer, little endian • W - 64 bit integer, big endian	May be followed by a “u” to indicate an unsigned value
Floating point	• q - 64 bit floating point, little endian • Q - 64 bit floating point, big endian	Supports IEEE floating point values only
Padding	• x - skip 1 byte of character padding • x<n> - to skip <n> number of bytes • x”*” - to ignore the remainder of the input. This option is only allowed if the padding is the last item in the format.	
Strings	• a - to indicate a string • a<n> - to specify a fixed length of <n> • a“*” - if an arbitrary length string must be deserialized. The “*” option is only allowed if the string is the last item in the format.	
IP address values	• Z - IP address, network order. Only allowed if the IP address is the last item in the format string. If there are greater than 16 bytes left in the input expression, an IPv6 address is read and if there are 16 bytes, an IPv4 address is read. Less than 16 bytes is read as a NULL value. • Z4 - IPv4 address, network order • Z6 - IPv6 address, network order. May also be used to deserialize an IPv4-mapped-IPv6 address. • z4 - IPv6 address, little endian	

Example:

This example gets the port and address from a string and puts them into corresponding local variables. It assumes a string input that contains 2 bytes of padding, a 2-byte port number, and a 4-byte IPv4 address, where both port and address are in network (big endian) order. The format string, `"x2SZ4"`, function contains:

- `x2` for 2 bytes of padding.
- `S` for a big-endian 16-bit integer.
- `Z4` for an IPv4 address.

If the `binary_data` input is smaller than 8 bytes, the `addr` result will be null; if the `binary_data` is smaller than 4 bytes then both `port` and `addr` will be null.

```
function "extract_port_and_address" (string "binary_data", \
                                     writable ipaddress "addr",\
                                     writable integer "port_number"){
    if true then \
      deserialize(input: binary_data, format: "x2SZ4", \
                  'output[0]': port_number, 'output[1]': addr)
}
```

2.6.11 Diameter

Diameter actions attempt to create and send a message to a remote Diameter peer. The actions have static built-in arguments and dynamic arguments. Diameter actions correspond to Diameter commands defined in a dictionary.

```
Diameter.<DictionaryName>.<CommandName>-<CommandSuffix>(ArgName1:arg_expression1,
ArgName2:arg_expression2, ...)
```

Where:

DictionaryName is the name of the dictionary that defines the command, for example, “Gy”.

CommandName is the name of the Diameter command from the dictionary, for example “CC”.

CommandSuffix is one of:

- Request sets the r-bit in the Diameter message header.
- Answer clears the r-bit in the Diameter message header.
- Answer-Error clears the r-bit in the Diameter message header and sets the e-bit in the Diameter message header.

Static arguments are:

Argument	Description	Type	Required (Yes/No)
Proxiable	Sets the p-bit in the outgoing Diameter message header. If omitted, or if the expression evaluates to null, the default value from the dictionary is used.	Boolean	No
Retransmitted	Sets the t-bit in the outgoing Diameter message header. If omitted, or if the expression evaluates to null, a default of false is used.	Boolean	No
End-to-End	Sets the end-to-end identifier in the outgoing Diameter message header. If omitted, or if the expression evaluates to null, an internally generated locally unique integer is used.	Integer	Yes for “-Answer” and “-Answer-Error” actions No for “-Request” actions
Hop-by-Hop	Sets the hop-by-hop identifier in the outgoing Diameter message header. If omitted, or if the expression evaluates to null, an internally generated locally unique integer is used.	Integer	Yes for “-Answer” and “-Answer-Error” actions

Argument	Description	Type	Required (Yes/No)
			No for “-Request” actions
Application-Id	Sets the application identifier in the outgoing Diameter message header. If omitted, or if the expression evaluates to null, the application id from the dictionary is used.	Integer	Yes for “-Answer” and “-Answer-Error” actions No for “-Request” actions
PeerIndex	Specifies the index of the peer to which to send the message. The value is obtained from the request PeerIndex field. When the argument is present, and the expression evaluates to null, the message is not sent.	Integer	Yes for “-Answer” and “-Answer-Error” actions No for “-Request” actions
Request-Origin-Host	Sets the origin host of the request for an outgoing answer. The value of this argument should be the origin host of the peer that sent the request for this answer. If present, the value of the expression is used along with the end-to-end ID when saving the answer in the pending answer queue. When absent, the answer is not saved in the pending answer queue.	String	No
Queued-For-Delivery	Serves as a return code for the action. When present, the expression is set to true if the message is added to a peer's outgoing queue for delivery, and set to false otherwise. A value of true does not necessarily mean the message was sent to the peer, only that the message was added to a peer's queue.	Boolean	No

Dynamic arguments are:

```
'<AvpName1>{[Index1]}{.<AvpName2>{[Index2] ...}'
```

Where:

- AvpName is the name of an attribute-value pair (AVP) from the dictionary for example, “Result-Code”
- Index is a numeric string. When Index is absent, an instance number of 0 is assumed.

The dynamic arguments are valid if all required arguments are present. Duplicate arguments cause SandScript reload to fail. If an AVP is required in a command, but the minimum number of instances is not present, SandScript will fail to reload. Optional dynamic arguments have a minimum instance count of 0 in the dictionary. For arguments that allow multiple instances, the index within the square brackets specifies the instance number. When more than one instance is used with an action, they must be used in the proper order starting with instance 0. So the user must specify instance 0 first, then 1, then 2 and so on. If multiple instances of an argument are not specified in the correct, order SandScript will fail to reload. The type of the argument value must be assignment-compatible with the type of the argument.

If an AVP name starts with anything other than a lower or upper case letter, or if the name contains anything other than alphanumeric characters or underscores, then the full argument name needs to be surrounded by single quotes. For example:

```
'Session-Id', 'Multiple-Services-Credit-Control{1}.Requested-Service-Unit'
```

Example:
This simple SandScript example sends a request when a flow starts. It assumes that a dictionary called “Gy” is configured:

```
if expr(Flow.IsNew) then \
    'Diameter.Gy.CC-Request' \
              ('Session-Id': “mysessionid”, \
              'Origin-Host':  PolicyEngine.Name, \
              'Origin-Realm': "sandvine.com", \
              'Destination-Realm': "sandvine.com", \
              'Auth-Application-Id': 4, \
              'User-Name': "James Bond", \
              'Event-Timestamp': Now, \
              'Service-Context-Id': 1, \
              'CC-Request-Type': 1, \
              'CC-Request-Number': 1, \
```

```
                'Service-Information.PS-Information.PDP-Address': 255, \
                'Multiple-Services-Credit-Control.Requested-Service-Unit': true, \
                'Multiple-Services-Credit-Control.Rating-Group': 129, \
                'Subscription-Id.Subscription-Id-Type': 130, \
                'Subscription-Id.Subscription-Id-Data': 131, \
              'Service-Information.PS-Information.X3GPP-GPRS-Negotiated-QoS-Profile': 132,
 \
                'Service-Information.PS-Information.SGSN-Address': 133, \
                'Service-Information.PS-Information.GGSN-Address': 134, \
                'Service-Information.PS-Information.Called-Station-Id': 135, \
                'Service-Information.PS-Information.X3GPP-SGSN-MCC-MNC': 136)
```

Example:
This SandScript responds to a request with a specific session ID:

```
if expr(Diameter.Gy.'CC-Request'.'Session-Id' = "mysessionid") then \
    Diameter.Gy.'CC-Answer' \
        (PeerIndex: Diameter.Gy.'CC-Request'.PeerIndex, \
        'End-to-End': Diameter.Gy.'CC-Request'.'End-to-End', \
        'Hop-by-Hop': Diameter.Gy.'CC-Request'.'Hop-by-Hop', \
        'Request-Origin-Host': Diameter.Gy.'CC-Request'.'Origin-Host', \
        'Result-Code': 2001, \
        'Origin-Host': PolicyEngine.Name, \
        'Origin-Realm': "sandvine.com", \
        'Auth-Application-Id': Diameter.Gy.'CC-Request'.'Auth-Application-Id', \
        'CC-Request-Type': Diameter.Gy.'CC-Request'.'CC-Request-Type', \
        'CC-Request-Number': Diameter.Gy.'CC-Request'.'CC-Request-Number', \
        'Session-Id': Diameter.Gy.'CC-Request'.'Session-Id', \
        'Application-Id': Diameter.Gy.'CC-Request'.'Application-Id')
```

2.6.12 Divert

Use the divert action to divert a flow to the specified destination.

```
divert destination "<name>" {from {client|server} interface "<interface1>" {from client|server
 interface "<interface2>"}}
```

Where:

name is the destination to divert to.

interfaceN is the destination interface to which to divert a particular direction of the flow. Interfaces may be specified on the divert action if the divert destination was defined with two interfaces.

Example:
For example:

```
destination "http_divert" divert ip 20.0.0.1 reset_server true tcp_syn true
if expr(Flow.SessionProtocol = Protocol.Http) \
   then divert destination "http_divert"
```

2.6.13 Event

Events allow one application to raise an event that other applications can listen for and take action on.

The event must be defined in SandScript before it can be used as an action.

```
Event.<Name>(<ArgName>:<arg_expression>, <ArgName>:<arg_expression>, ...).IsComplete
```

Where:

<Name>	is the name of the event as provided in the definition.
<ArgName>	is the name of an event argument from the definition.
<arg_expression>	is assigned to the corresponding ArgName argument when the event is raised. The expression must be assignment compatible with the type of the argument.
IsComplete	is a boolean that indicates that the event associated with it is completed with a successful or failed status. This is an optional argument. Use it to gather information about the event status and different attribute values, accordingly.

Example:
This example raises an event on every new flow. When the event is run, a row is inserted in the table. If the flow was an HTTP flow, the string HTTP is stored in the table, otherwise NULL is stored in the table. The remainder of the SandScript evaluation happens before the event is executed, but there is no guarantee on how soon after the SandScript evaluation finishes that the event will be run.

```
table "ExampleTable" (integer "key", string "value") timeout none \
unique by (Table.ExampleTable.Key) \
event "ExampleEvent" (integer "IntegerArg", string "StringArg")

PolicyGroup expr(Flow.IsNew){
  if expr(Flow.SessionProtocol = Protocol.Http) then \
    Event.ExampleEvent(IntegerArg: Flow, StringArg: "HTTP")
  if true then Event.ExampleEvent(IntegerArg: Flow)
}

if expr(Event.ExampleEvent) then \
  set ExampleTable[Event.ExampleEvent.IntegerArg].Value = \
  Event.ExampleEvent.StringArg
if expr(Flow.IsEnd) then delete_row Table.ExampleTable[Flow]
```

2.6.14 HTTP Response

The http_response action provides the ability to respond to a host HTTP request with a customized response.

```
http_response <response_string>
```

Where: response_string is a SandScript expression that returns the string with which to respond to the client HTTP request.

http_response is transaction aware, meaning the response will only be used in the transaction for which the conditions were matched. When used in conjunction with the analyze SandScript action, http_response can be applied to subsequent transactions within persistent and pipelined HTTP flows.

These actions occur when the http_response action is executed:

- The transaction response is replaced by the string provided to the http_response action.
- Responses to subsequent transactions in this actioned flow are dropped. Therefore, if http_response acts on transaction *i*, then the client will see no responses to transactions > *i*, and it will be forced to start a new flow with the server. Responses to previous transactions are left unmodified.
- The actioned packet has its FIN flag set.
- The PTS sends a RST packet to the server.

Note:
In all cases, you need to set `Flow.Stream.ReadOnly = false`, to allow the flow to be modified by the PTS. A flow starts as read-only in its first policy run, but if at any point the `Flow.Stream.ReadOnly` is not set to 'false', the flow will become read-only forever. When the flow is not read-only, the PTS buffers packets to reassemble the stream to analyze

HTTP. If a flow continues in non read-only mode, latency is added to the HTTP connection. This is a problem in case of lost or reordered packets. Set the `Flow.Stream.ReadOnly` value to "true" in the SandScript as soon as possible.

Example:
This first example sends an `http_response` to the HTTP flow's first transaction request:

```
if expr(Flow.SessionProtocol = Protocol.Http and Flow.IsNew) then \
 set Flow.Stream.ReadOnly = false and \
 http_response ("HTTP/1.1 307 Temporary Redirect\r\nLocation: \
     http://example.net\r\n\r\n")
```

Example:
This example replaces a 404 server response:

```
PolicyGroup expr(Flow.SessionProtocol = Protocol.Http) all {
 if true then set Flow.Stream.ReadOnly = false
 PolicyGroup
 {
  if expr(Flow.Server.Stream.Http.StatusCode = 404) then \
   http_response ("HTTP/1.1 <insert custom response here>")
  if true then analyze analyzer "http"
 }
}
```

Example:
This example illustrates an `http_response` action applied only to HTTP flows with a request “host” field containing captiveportalme.com and who have not paid for access to that host.

```
# Attribute to check if the user paid for access to "captiveportalme.com".
# You set the attribute to true on the SPB when the user
# pays and the attribute value change would be pushed to the PTS so the
# user is given access to the paid service.
attribute "paid_for_service" values "true" "false"

# now only redirect if the user hasn't paid
PolicyGroup expr(Flow.SessionProtocol = Protocol.Http) all {
 if true then set Flow.Stream.ReadOnly = false

    # Ensure we analyze until at least the end of the 1st transaction's request
 if expr(Flow.Client.Stream.Http.Command is null) then analyze analyzer "http"

 if expr(not(Flow.Client.Attribute.paid_for_service = "true") and \
  contains("captiveportalme.com", Flow.Client.Stream.Http.Host)) \
  then http_response (concat("HTTP/1.1 307 Temporary Redirect\r\nLocation:\
  http://www.mydomain.com:", Flow.Client.Stream.Http.Host))
}
```

Offline Mode Behavior

This action behaves the same in inline and offline modes, except that in offline mode, the modified response packet and server RST packet are sent out the service interface and the original server response cannot be withheld from the client.

Limitations

The PTS is unable to conduct an http_response action on transactions in which the HTTP response header spans multiple packets. This scenario occurs in less than 0.5% of HTTP flows.

To work-around this limitation, use the `Flow.Server.Stream.Http.CanHttpResponse` SandScript field to notify the SandScript writer that a transaction meets the criteria for this limitation so that alternate actions can be taken. (Refer to the *PTS SandScript Reference* for more information on SandScript fields.)

The PTS is unable to conduct an http_response action on transactions in which the http_response payload is too large to fit in a packet with a size equal to the MTU. With a realistic HTTP response payload, this can only occur with a pipelined transaction whose header just fits at the end of the packet after the previous transaction body. Studies indicate that on a wireless network,

less than 1% of flows are pipelined (on wired networks, pipelined flows are far less likely) and that of these flows, a very small percentage will be spaced such that the HTTP response payload will not fit in the space remaining up to the MTU.

To work-around this limitation, use the `Flow.Server.Stream.Http.FailedHttpResponse` SandScript field to notify the SandScript writer of transactions that meet this criteria so that these transactions can be countered differently, if desired.

2.6.15 Increment

The increment action increments the value of the specified SandScript field.

Increment is only compatible with writeable integer fields. When incrementing a histogram measurement, the histogram bin is specified using the "bin" syntax. By default, the value of the field is incremented by 1. The optional "by" syntax can be used to increment the value by some other value. When it is used to increment the value of a subscriber attribute, the attribute expiry time can be set using the "for" syntax.

```
increment <field> {bin <bin_expression>}{by <expression>}{for <time>}
```

Where:

field — Any writeable integer field.

bin_expression — An integer expression that defines the histogram bin to increment. Can only be used if `field` is a histogram measurement.

expression — An integer expression that defines the value to increment by. If the expression evaluates to null, the `field` will not be changed (it is not set to null).

time — An integer expression plus a unit of time, for example, “10 minutes” or “LocalTime(2010,01,01) - Now seconds”. Specifies when a subscriber attribute incremented by the action will expire and become null.

Example:
This measurement counts the total number of bytes seen on the element. It is updated at the end of every flow.

```
measurement "ExampleMeasurement"
if (Flow.IsEnd) then increment Measurement.ExampleMeasurement by Flow.TotalBytes
```

2.6.16 Mark

Marking actions are used to mark the flow using the ToS (type of service) field found in the IPv4 header and Traffic Class field in the IPv6 header. This marking is commonly used to indicate traffic priority to downstream devices.

Both mask and value are single bytes set into either the IPv4 ToS field or IPv6 Traffic Class. The dscp variant is a convenience for setting the upper 6 bits of this field; hence, the valid range of values is 0 to 63.

```
mark [divert|port <internal|external|subscriber|internet>] dscp <value 0..63>
mark [divert|port <internal|external|subscriber|internet>] mask <mask 0..255> <value 0..255>
```

Parameter	Description
dscp	Applies a Diffserv marking according to RFC 2474. Note that there is no mask with dscp syntax.
divert	Marks packets going towards the divert host when a flow is diverted.

Parameter	Description
port internal|external| subscriber|internet	Applies this mark only to packets destined to the port(s) of the specified side of the PTS. Internal is equivalent to subscriber and external is equivalent to internet.
mask	Specifies which bits of the TOS field to modify. Range is 0-255. Use 255 to modify all bits. For *mark dscp* this value is implied to be 252.
value	Specifies the octet to write to the IPv4 TOS field. For *mark dscp* this value may be 0-63 and is applied to the upper 6 bits of the TOS field.

Example:
These examples mark specific protocol traffic:

```
if expr(Flow.ApplicationProtocol = Protocol.eDonkey) then mark 255 10
if expr(Flow.ApplicationProtocol = Protocol.Bittorrent) then mark dscp 62
```

2.6.17 Record Decrement

Decrements the values of multiple members in a record local variable. Supported for integer members only.

```
<RecordVariable>.Decrement(<MemeberName1>:<expression>, <MemeberName2>:<expression>, ...)
```

Example:
For example:

```
record “ExampleRecord” (integer “intMember” default 5, ipaddress “addressMember”)
Record.ExampleRecord “RecordVariable”

# Decrease the integer member of the variable by 7.
if true then RecordVariable.Decrement(intMember: 7)
```

2.6.18 Record Increment

Increments the values of multiple members in a record local variable. Supported for integer members only.

```
<RecordVariable>.Increment(<MemeberName1>:<expression>, <MemeberName2>:<expression>, ...)
```

Example:
For example:

```
record “ExampleRecord” (integer “intMember” default 5, ipaddress “addressMember”)
Record.ExampleRecord “RecordVariable”

#Increase the integer member of the variable by 5.
if true then RecordVariable.Increment(intMember: 5)
```

2.6.19 Record Reset

Sets the values of all members in a record local variable to their default values.

```
<RecordVariable>.Reset()
```

Example:

```
record “ExampleRecord” (integer “intMember” default 5, ipaddress “addressMember”)
Record.ExampleRecord “RecordVariable”

#Set the integer member of the variable to 5 (the default) and the address member to null
if true then RecordVariable.Reset()
```

2.6.20 Record Set

Sets the values of multiple members in a record local variable. The order in which members are set is not guaranteed. If order is important, multiple set actions should be used instead.

```
<RecordVariable>.Set((<MemeberName1>:<expression>, <MemeberName2>:<expression>, ...)
```

Example:
For example:

```
record “ExampleRecord” (integer “intMember” default 5, ipaddress “addressMember”)
Record.ExampleRecord “RecordVariable”

# Set both members of the variable.
if true then RecordVariable.Set(addressMember: IPAddress(“192.168.2.1”), intMember: 10)
```

2.6.21 Reevaluate

The reevaluate action causes SandScript policy to be reapplied to a flow. Note that all flows are periodically reevaluated. This action prioritizes certain flows for reevaluation. Do not overuse this action.

```
reevaluate flow <triggering_conditions>
```

Where triggering_conditions is the event that causes the reevaluation.

2.6.21.1 Reevaluate flow after some time

```
if <condition> then reevaluate flow after <n> seconds|minutes|hours
```

Where:

- Only one timer can be armed per flow.
- If multiple reevaluation intervals are specified for a flow, the smallest is used.
- Timers are one shot. They are rearmed for each evaluation.
- There is no protection against setting too many timers with short intervals. For example, every flow could have a one second timer, which would not be sustainable for a million flows.

2.6.21.2 Reevaluate flow on DataTransferDirection

This action causes SandScript policy to be reapplied immediately to the flow when the data transfer direction changes (for example, from bidirectional to unidirectional). This action is typically used in conjunction with Session Management rules to quickly detect and limit uploads.

```
if <condition> then reevaluate flow on DataTransferDirection
```

Example:

For example:

```
if expr(Flow.IsApplicationProtocol("bittorrent","edonkey") then \
            reevaluate flow on DataTransferDirection
PolicyGroup {
 PolicyGroup expr(Flow.IsApplicationProtocol("edonkey") {
  PolicyGroup receiver class "external" {
   if expr(Flow.IsReevaluatingPolicy and \
    Limiter.AllUpload.Priority2Existing) then tcp_reset
   if expr(not(Flow.IsReevaluatingPolicy) and \
    Limiter.AllUpload.Priority2New) then tcp_reset
  }
 }
 expr(Flow.IsApplicationProtocol("bittorrent",) {
  PolicyGroup transfer uni and receiver class "external" {
   if expr(Flow.IsReevaluatingPolicy and \
    Limiter.AllUpload.Priority3Existing) then tcp_reset
   if expr(not(Flow.IsReevaluatingPolicy) and \
    Limiter.AllUpload.Priority3New) then tcp_reset
  }
 }
}
```

Note that this example uses a limiter that is defined elsewhere.

2.6.21.3 Reevaluate IP

The reevaluate IP action causes SandScript policy to be reapplied to all flows and/or the subscriber for a specified IP address.

Note that all flows and subscribers are periodically reevaluated. This action prioritizes certain flows and subscribers for reevaluation. Be careful not to overuse this action.

```
reevaluate_ip(address:<expression> \
    {, flows: true|false} \
    {, subscriber: true|false} \
    {, flows_first: true|false})
```

Where:

expression	Specifies the IP address to be scheduled.
flows	Indicates if all flows for the specified IP address at the time of the call will be scheduled for reevaluation (default true).
subscriber	Indicates if the subscriber associated with the specified IP address at the time of the call will be scheduled for reevaluation (default true).
flows_first	Indicates if the flows will be evaluated prior to the subscriber, otherwise the subscriber will be evaluated prior to the flows (default true).

Example:
To evaluate all the flows, as well as the subscriber, for 10.10.10.1 with the flows being evaluated first:

```
reevaluate_ip(StringToIP("10.10.10.1"))
reevaluate_ip(StringToIp("10.10.10.1"), flows: true, subscriber: true, flows_first: true)
```

To evaluate only the flows for the IP address 10.10.10.1:

```
reevaluate_ip(StringToIP("10.10.10.1"), flows: true, subscriber: false)
```

To evaluate only the subscriber for the IP address 10.10.10.1:

```
reevaluate_ip(StringToIP("10.10.10.1"),flows: false, subscriber: true)
```

To evaluate the subscriber followed by flows for the IP address 10.10.10.1:

```
reevaluate_ip(StringToIP("10.10.10.1"), flows: true, subscriber: true, flows_first: false)
```

2.6.22 Set

Use the set action to set SandScript fields that support modification.

```
set <fieldname> = <expression> {for <timeout>}
```

Where timeout is an integer expression plus a unit of time, for example, “10 minutes” or “LocalTime(2010,01,01) - Now seconds”. Specifies when the expression will expire (and become null).

Example:
For example:

```
set flow.client.attribute.x = 1
set flow.server.attribute.some_attr = NULL
```

2.6.23 Shape

A shaping action references a shaper and the node-qualifier to shape.

```
shape from|to client|server|sender|receiver|subscriber|internet|source|destination
shaper "<Shaper_name>" {priority "<Priority_name>"} {channel "<Channel_name>"}
{unique by <expr>} {shared by <expr>}
```

Where:

from|to — specifies whether packets to or from the node qualifier that follows should be shaped. The two syntaxes will lead to different results when used with diverting.

client|server|sender|receiver|subscriber|internet|source|destination — indicates the direction in which packets will be shaped. For example, shape to client indicates that only packets travelling from server to client will be shaped.

<Shaper_name> — is the shaper to use.

<Priority_name> — if used, must correspond to a name specified in a shaper definition, otherwise, the priority clause must not be present in the action. If the specified priority does not exist, an error will be generated.

<Channel_name> — if used, must correspond to a name specified in a shaper definition, otherwise, the channel clause must not be present in the action. If the specified channel does not exist, an error will be generated.

unique by and shared by — determine whether the bandwidth allocated by the shaper is allocated totally to the user instance or is shared across all instances. Where:

- expr is one of:

  ```
  client|server-(ip|sub|port|class)
  transport
  protocol
  any expression
  ```

- If ip is specified, a queue is created for each IP address. If sub is specified, a single queue is created for each subscriber, even if the subscriber has more than one IP address.

- If sub is specified, the PTS must be subscriber aware (subscriber-IP mapping must be in place) or it is ignored.
- SandScript expressions are recommended.
- To do shaping unique by subscriber sessions, use the expression `Session.Id`. Do not use unique-by IP address.

Up to four shapers can simultaneously shape a flow. Alarm model 23: Policy error, is triggered if SandScript exceeds this limit.

Example:
This example shapes HTTP download traffic (and upload to internal servers) to an aggregate total of 20Mbps. Note that the allocation of bandwidth to each client is fair. All active clients share the available bandwidth equitably. In this example, a shaper called fair is defined and then used in the following shaping rules.

```
shaper "Fair" 20Mbps depth 8Kbytes
if expr(Flow.ApplicationProtocol = Protocol.HTTP) then shape \
 to subscriber shaper "Fair" shared by \
 Flow.Subscriber.IpAddress
```

2.6.24 TCP Reset

The tcp_reset action resets all flows that match a specific condition. This is a licensed feature.

```
if <condition> then tcp_reset
```

Example:
For example:

```
# all gnutella uploads
if protocol "gnutella" and receiver class "external" then tcp_reset
```

2.6.25 Tee

Once a tee destination has been defined, the tee action is used to tee packets to the destination(s).

```
if <condition> then tee from|to
client|server|sender|receiver|subscriber|internet|source|destination destination "<Name>"
```

Where:

from|to — specifies whether packets to (transmitted by the PTS) or from (received by the PTS) the node qualifier that follows should be teed. The two syntaxes will lead to different results when used with diverting.

client|server|sender|receiver|subscriber|internet|source|destination — indicates the direction in which packets will be teed. For example, tee to client indicates that only packets travelling from server to client will be teed.

destination “<Name>” — is the name of the destination to tee the packets to.

For each flow:

- there can be a maximum of four unique destinations per flow (file and tee to a device)
- for each destination, there can be a maximum of four node qualifiers, although using two node qualifiers is most common.

Example:
For example:

```
if protocol "unknownTCP"  then \
 tee from client destination "destFile1" and \
 tee from server destination "destFile1"
```

2.6.26 Compatible Actions

Some actions are incompatible; they cannot both be applied at the same time. Actions which do not appear in this table are always compatible with all other actions. X indicates the actions are incompatible. * indicates an action that may be incompatible with itself.

	allow	block	continue	count	count demo-graphic	count sub-scriber	divert	http_ response	limit/ tcp_reset	mark	shape	tee
allow	X	X	X				X	X	X		X	
block	X	X	X	X	X	X	X	X	X	X	X	X
continue	X	X	X				X	X	X		X	
count		X		X								
count demo-graphic		X			X							
count sub-scriber		X				*						
divert	X	X	X				X	X	X			
http_ response	X	X	X				X	X	X		X	
limit/ tcp_reset	X	X	X				X	X	X			
mark		X						X		*		
shape	X	X	X					X				
tee												*

- Multiple mark actions are allowed if they apply to different ports.
- Multiple count subscriber actions are allowed if they apply to different node qualifiers.
- Multiple shape actions are always allowed, up to a maximum of four per flow.
- Multiple tee actions are allowed, up to a maximum of two destinations per flow. Teeing multiple times to the same destination with the same node qualifier will result in only a single copy of the packet being sent to the destination.

The first action that is matched is applied, and if a subsequent action is incompatible with the first, it is not applied. For example, divert and tcp_reset are incompatible actions. If both actions are indicated in the same SandScript file with this order of precedence:

```
if protocol "bittorrent" then divert
if protocol "bittorrent" then tcp_reset
```

then all BitTorrent flows will get diverted as the divert action appears first in the SandScript file.

3

SandScript Definitions

3.1 Local Variables

Local variables are used to store the values of frequently used complex expressions (a complex expression is any expression that uses one or more operators or functions).

By storing the result of such an expression, the value can subsequently be retrieved quickly using the local variable instead of evaluating the more expensive expression multiple times. Local variables can be initialized to a value when they are defined and can be set/reset any number of times.

Local variables must be defined within a PolicyGroup. Typically, variables only have scope within the containing PolicyGroup, unless they are defined as shared. The scope of shared variables extends until the end of SandScript. Shared local variables defined in a PolicyGroup with a context different from that in which SandScript is being run are available but initialized to null.

Local variables never retain their value across SandScript evaluations. The value of a local variable is always reset to its initial value, or null if one is not specified, when it comes into scope. Local variables are defined as:

```
{shared} <type> "<Name>" {= <initial_value>}
```

Where:

shared Optional. Extends the variable's scope until the end of SandScript.

type Type of the local variable. Supported types include: string, float, integer, boolean, ipaddress, Table.TableName:cursor (a table row), Session:cursor (an IP indexed session), Record.<name>.

Name Name of the local variable. Note that the name:

- Must begin with a letter and can only contain letters, digits, and underscores.
- Must not be a reserved SandScript keyword.
- Is not case sensitive.
- Cannot conflict with other local variables that are still in scope.

The name is used to lookup or set the value of the local variable. In the case of a cursor local variable, the name is used to:

- Obtain column values of a particular table row (for example, VarName.ColumnName).
- Access SandScript fields of a particular IP indexed session (for example VarName.IsMapped).

initial_value A complex expression which is assigned to the local variable. The expression must be assignment compatible with the type of the local variable. If a local variable is not assigned an initial value, it will remain null until it is assigned a value using the set action.

Example:
For example:

```
PolicyGroup all {
    integer "var1"
    string "var2" = "Value"
    boolean "var3" = expression
    shared integer "sharedVar" = 180
    if condition(s) then set var1 = expression
    if condition(s) then set expression = var2
    if expr(var3) then action(s)
}
table "MyTable" (integer "key", integer "value") timeout none \
    unique by (Table.MyTable.Key)
PolicyGroup all {
    Table.MyTable:cursor "row1"
    Table.MyTable:cursor "row2" = Table.MyTable[2]
    Table.MyTable:cursor "row3" = Table.MyTable.Create(2)
    if condition(s) then row2.Set(value: 7)
```

```
        if condition(s) then set row3.value = 180
        if condition(s) then set row1 = row2
        if expr(row2 = row3) then action(s)
        if expr(row1.value > 10) then actions(s)
    }
```

3.2 Classifiers

Classifiers are used to apply labels to SandScript contexts such as flows or subscribers and to categorize flows or subscribers into meaningful groups.

The set action is used to set the classifier's value through its SandScript field. The SandScript field can be set any number of times in SandScript and can be used by the classifier to: apply a condition, measure, limit, or shape unique-by. The Flow.Classifier.Name SandScript field can only be used in rules and classifications that apply to flows. Use Classifier.Name when using the classifier on subscribers.

Classifiers are defined either by type or listed explicitly. The syntax of classifier instances specified by type is:

```
classifier "<Name>" type
string|integer|integer8|integer16|integer32|unsigned8|unsigned16|unsigned32
```

If defined this way, the classifier can be set to any SandScript field of the appropriate type. You can limit the range of input values by using one of the smaller types. If you try to assign a value larger than the type to the classifier, the value is truncated to fit within the classifier.

The syntax of classifier instances listed explicitly is:

```
classifier "<Name>" values "<value1>" {"<value2>" …}
```

Where:

- <Name> - Name is the name of the classifier. This is used to identify the classifier in the CLI and when referencing the classifier using SandScript fields.
- "value1" {"value2" …} - This is a list of classifier values. These values are used as the names of the constant SandScript fields for use to set the value of the classifier (Classifier.Name.Value).

Classifier names and values must follow these rules:

- Must begin with a letter and can only contain letters, digits, and underscores.
- Not case sensitive.
- Names must be unique amongst names and values must be unique amongst values for the given classifier (different classifiers can have the same values).

Each classifier begins each SandScript evaluation without a value (the Classifier.Name SandScript field is null). A classification does NOT persist across SandScript evaluations. The value is reassigned each time SandScript is evaluated. If none of the classification conditions are matched, the classifier will remain without a value (the SandScript field will remain null).

The maximum number of subscriber classifiers that can be used is 76. Of that number, 64 can be a sum and 12 can be used to report the maximum value over the past 12 hours.

Example:
This example classifies traffic into protocol categories, measures and publishes the number of bytes transferred per category per subscriber.

```
classifier "Category" values "Web" "P2P" "Other"
PolicyGroup {
  if expr(Flow.SessionProtocol = Protocol.Http) then \
    set Flow.Classifier.Category = Classifier.Category.Web
  if expr(Flow.IsApplicationProtocol("ares","bittorrent","edonkey","gnutella")) then \
```

```
    set Flow.Classifier.Category = Classifier.Category.P2P
  if true then \
    set Flow.Classifier.Category = Classifier.Category.Other
}

measurement "BytesPerCategoryPerSub" sum(Flow.Subscriber.SubscriberBytes) \
  over publish_interval unique by (Subscriber, Flow.Classifier.Category)
publish "BytesPerCategoryPerSub" Measurement.BytesPerCategoryPerSub
```

Example:
This example measures and publishes the number of bytes transmitted and received per protocol for each CMTS and QAM (CMTS and QAM are subscriber attributes):

```
attribute "CMTS" type string
attribute "QAM" type string
classifier "CMTS" type string
classifier "QAM" type string
if true then \
    set Flow.Classifier.CMTS = Flow.Subscriber.Attribute.CMTS and \
    set Flow.Classifier.QAM = Flow.Subscriber.Attribute.QAM

measurement "DownstreamBytesPerProtocolPerCmtsQam" \
  sum(Flow.Subscriber.Rx.ProtocolBytes) \
  over publish_interval \
  unique by (Flow.ApplicationProtocol.StatsProtocol,
             Flow.Classifier.CMTS,
             Flow.Classifier.QAM)
publish "DownstreamBytesPerProtocolPerCmtsQam" \
  Measurement.DownstreamBytesPerProtocolPerCmtsQam
measurement "UpstreamBytesPerProtocolPerCmtsQam" \
  sum(Flow.Subscriber.Tx.ProtocolBytes) \
  over publish_interval \
  unique by (Flow.ApplicationProtocol.StatsProtocol,
             Flow.Classifier.CMTS,
             Flow.Classifier.QAM)
publish "UpstreamBytesPerProtocolPerCmtsQam" \
  Measurement.UpstreamBytesPerProtocolPerCmtsQam
```

3.3 Measurements

3.3.1 Measurements Overview

Measurements are declared through SandScript to collect traffic and subscriber-related statistics.

Measurements are taken in the order that they are defined in SandScript, however, they are taken after all other applicable actions have been run. This does not apply to generic measurements which are modified explicitly via the set/increment/decrement actions.

This example measures connections if the flow is blocked:

```
measurement "blockedConnections" connections where expr(Flow.IsBlocked)
if expr(Flow.ApplicationProtocol = Protocol.ares) then block
```

A maximum of 1000 measurements can be defined in SandScript. The measurement types that are supported are:

Measurement Type	Description
Generic (user-defined)	The value of this type of measurement is controlled explicitly through SandScript using the set, increment, or decrement actions. It can be used to implement any kind of user-defined counter or statistic.
Active Connections	The current number of active connections that match a set of conditions.
Bitrate	The current amount of bandwidth associated with traffic that matches a set of conditions.
Hosts	The number of active hosts that match a set of conditions.
Expression	A measurement can be used to expose the value of a SandScript expression.
Top	Sums the value of an expression over time and finds the top instances.
Sum	Aggregates the value of a SandScript expression over time.
Histogram	Counts observed values into bins.

Measurement values are available using the CLI command `show policy measurements`, so you can monitor the values in real-time.

Measurement values are also available in SandScript via SandScript fields and can be published and reported on in NDS, allowing a number of custom user-defined statistics and reports to be defined. For detailed information on SandScript fields, refer to the *PTS SandScript Reference*.

3.3.1.1 Measurements

Collects traffic and subscriber related data.

```
measurement "<Name>" <type> {over <period>} {where <condition(s)>} {unique by <unique_by_spec>}
 {id <index>}
```

Name

The name of the measurement, used to identify it through the CLI and SandScript. The name must:

- Begin with a letter.
- Only contain letters, digits, and underscores.
- Not use case sensitivity.
- Be unique amongst measurements and limiters

Note:
You cannot have a measurement named “Protocols” and a limiter named “Protocols”; unique names are required.

type

Specifies the measurement type. If no type is specified, it is a generic measurement. Otherwise, measurements come in these types:

```
connections
bitrate
hosts
sum
top
expr
histogram
```

Period

The exact behavior depends on the measurement type. In general:

- Peak values are reset at the start of each period.
- With the exception of connection measurements, the set of unique instances being measured by unique-by measurements is cleared at the start of each period.
- Can be specified as a time in seconds, minutes, or hours (for example, “over 60 seconds”, “over 5 minutes”, “over 1 hour”).
- For generic and sum measurements, if unique_by_spec is specified as between 1 and 3 enumerated classifiers, you can specify the period as infinity. You can publish the measurement though its values never reset.
- You can specify `publish_interval` to synchronize it with the statistics publishing interval.
- You MUST synchronize unique-by measurements with the publishing interval to publish.
- Required for measurements where a unique-by_spec is specified, optional for all others. The default behavior is to never reset the value and/or peak values of the measurement.

Condition(s)

The conditions(s) that indicate what to measure. The format of the condition(s) is the same as it is for rules. The condition is optional; the default value is **true**, and all traffic or hosts (depending on the measurement type) are measured.

unique_by_spec

When this measurement is specified, it measures a value for each instance as defined in the specification. These values are acceptable":

```
client-ip|server-ip|internal-ip|receiver-ip|sender-ip|client-subscriber|server-subscriber|subscriber
```

For example, “unique by client-ip”.

Or, can be a SandScript expression list (a comma separated list of fields or expressions inside parenthesis). For example:

```
unique by (Flow.Subscriber.Attribute.CMTS, Flow.Subscriber.Attribute.QAM)
unique by (Subscriber, Flow.Classifier.APN)
```

The total number of unique instances being measured across all measurements is limited according to platform:

- PTS 24000 supports 63 million.
- PTS 22000 supports 20 million.
- PTS 14000 supports 20 million.

pt

An alarm is raised when the limit is exceeded. A published measurement can be unique by no more than four unique-by classifiers, or no more than one protocol and three classifiers.

index

A static identifier for the measurement that persists through reloads and restarts. Use this identifier to poll the measurement via SNMP in the SANDVINE-MIB::svMeasurementsTable.

3.3.1.2 Generic Measurement

User-defined measurements are referred to as generic measurements.

```
measurement “<Name>” {over <period>} {unique by <unique_by_spec>} {aggregate sum|max}
```

aggregate {sum|max} is the aggregation type of the measurement. The aggregation types are:

- sum, (the default) the measurement is treated as a counter when its values are aggregated across modules and over time intervals in reports.
- max, the measurement is treated as a gauge where the peak value of the published expression is taken, across modules and over time intervals in reports.

Unlike other measurement types, simply defining the measurement does not provide any value. Policy rule(s) must be written to set the `Measurement.Name`, or `Measurement.Name.Current` SandScript fields which are initialized to 0. Setting generic measurements to NULL is equivalent to setting them to 0.

The measurement may or may not be over an interval. If it is not over an interval, then you can set the value of the defined measurement using the `Measurement.Name` SandScript field. If the measurement is over an interval, then the values of the measurement that are exposed to policy are:

- `Measurement.Name` for the value of the measurement in the previous interval.
- `Measurement.Name.Current` for the value of the measurement in the current interval.

Note:
For measurements over an interval, Sandvine recommends modifying the `Measurement.Name.Current` SandScript field. While the policy that sets, increments, or decrements the `Measurement.Name` SandScript field for measurements over an interval is valid, it has the same effect as performing the action on the `Measurement.Name.Current` SandScript field.

Example:
To set the value of this measurement over an interval using the set, increment, or decrement actions, use the `Measurement.<Name>.Current` SandScript field. For example:

```
measurement "NewHTTPFlows" over publish_interval
if expr(Flow.IsNew and Flow.ApplicationProtocol = Protocol.HTTP)\
then increment Measurement.NewHTTPFlows.Current
```

3.3.1.3 Active Connection Measurement

This type of measurement can be used to measure the instantaneous number of active connections that match a set of conditions.

Syntax:

```
measurement "<Name>" connections {over <period>} {where <condition(s)>}
unique by <unique_by_spec>}
```

For connection measurements, period serves only to reset the peak values of the measurement.

A flow can only be measured by four different connection measurements at a time. If this limit is exceeded, an alarm will be raised. For example, with this policy:

```
measurement "AllConnections" connections
measurement "InternalClientConnections" connections where client class "internal"
measurement "TcpConnections" connections where layer4protocol TCP
measurement "HttpConnections" connections where expr(Flow.ApplicationProtocol = Protocol.HTTP)
measurement "AllConnectionsPerSub" connections unique by subscriber
```

An HTTP flow with a client in the "internal" network class will match all five conditions. These flows will be measured by the first four measurements and NOT by the fifth and an alarm will be raised.

This example measures the current active number of HTTP connections being shaped:

```
measurement "ShapedHttpConnections" connections where \
protocol expr(Flow.ApplicationProtocol = Protocol.HTTP) \
and expr(Flow.IsShaped)
```

This example measures the current active number of BitTorrent unidirectional uploads per subscriber. The peak values are reset at the start of each publishing interval.

```
measurement "BittorrentUniUpPerSub" over publish_interval where \
expr(Flow.ApplicationProtocol = Protocol.BitTorrent) \
and transfer uni and sender class "internal" unique by (Subscriber)
```

3.3.1.4 Bitrate Measurement

Bitrate measurements measure the instantaneous bandwidth of traffic that matches a set of conditions.

Syntax:

```
measurement "<Name>" bitrate {to|from sender|receiver|client|server|internet|subscriber}
{over <period>}{where <condition(s)>}
```

The connections to measure are selected by specifying the appropriate condition. Note that in inline mode the bitrate is calculated on egress, after blocking and shaping (anything that can drop or modify a packet). In offline mode, it is calculated on ingress.

The direction to measure is specified in these ways:

- By specifying to|from and a node qualifier (client, server, sender, receiver, subscriber or internet). This approach should not be used to separate upstream from downstream traffic since, in the case the client is internal, “from client” measures upstream bandwidth and in the case that the server is internal, “from client” measures downstream bandwidth - so a “from client” measurement will measure a mix of upstream and downstream.

  ```
  measurement "HttpFromClient" bitrate from client \
  where expr(Flow.ApplicationProtocol = Protocol.HTTP)
  ```

- By using the source and destination node qualifiers in the condition. These are special node qualifiers that can only be used in bitrate measurement definitions. They are used to indicate the direction of the packets that should be measured, for example, measure all packets coming from the “internal” network class.

  ```
  measurement "HttpUpload" bitrate \
  where expr(Flow.ApplicationProtocol = Protocol.HTTP) and source class "internal"
  ```

These techniques cannot be combined (specifying from/to and source/destination in the condition will not load). If neither one are specified, all packets are measured.

For bitrate measurements, period serves only to reset the peak value of the measurement.

A flow can only be measured by four different bitrate measurements at a time. If this limit is exceeded, an alarm will be raised. For example, with this policy:

```
measurement "AllBitrate" bitrate
measurement "UpstreamBitrate" bitrate where source class "internal"
measurement "DownstreamBitrate" bitrate where destination class "internal"
measurement "HttpFromClient" bitrate from client \
where expr(Flow.ApplicationProtocol = Protocol.HTTP)
measurement "HttpFromServer" bitrate from server \
where expr(Flow.ApplicationProtocol = Protocol.HTTP)
```

All HTTP flows will match all five conditions. These flows will be measured by the first four measurements and NOT by the fifth and an alarm will be raised.

Example:
This example measures all upstream bandwidth:

```
measurement "Upstream" bitrate where source class "internal"
```

Example:
This example measures all downstream HTTP bandwidth:

```
measurement "DownstreamHttp" bitrate \
where expr(Flow.ApplicationProtocol = Protocol.HTTP) and destination class "internal"
```

Example:
This example measures all upstream HTTP bandwidth for subscribers in the "gold" tier:

```
measurement "UpstreamHttpGold" bitrate \
where expr(Flow.ApplicationProtocol = Protocol.HTTP \
and Flow.Subscriber.Attribute.Tier = "gold") \
and source class "internal"
```

3.3.1.5 Host Measurement

Host measurements measure the number of hosts that match a set of conditions.

Syntax:

```
measurement "<Name>" hosts {over <period>} {where <condition(s)>}
```

The condition is restricted to those that apply to subscribers. Flow conditions (for example, protocol or SandScript expressions involving flow fields) cannot be used and will not load.

Host measurements only measure hosts if subscriber classes are defined in policy (subscriber-ip mapping is enabled). Only IP addresses in a subscriber class are measured. For host measurements, period serves only to reset the peak value of the measurement.

Example:
This example measures the number of hosts with the “IsSpammer” attribute set to “true”.

```
attribute "IsSpammer" type boolean
measurement "Spammers" hosts where expr(Subscriber.Attribute.IsSpammer)
```

3.3.1.6 Sum Measurement

This type of measurement is used to aggregate the value of an expression over time. Typically sum measurements are used to count bytes or packets.

Syntax:

```
measurement “<Name>” sum(<expression>) {over <period>} {where <condition(s)>}
{unique by <unique_by_spec>}
```

The unique_by_spec is optional, if specified, period must also be specified.

If no unique_by_spec is specified, period controls how often the measurement value is reset to 0. If a unique_by_spec is specified, period controls how often the instances are flushed. In both cases, the peak values are also reset at the start of each period.

This example counts the total number of bytes for all HTTP connections. The value of the measurement is reset at the start of each publishing interval.

```
measurement "HttpBytes" sum(Flow.Tx.ProtocolBytes) over publish_interval\
where expr(Flow.ApplicationProtocol = Protocol.HTTP)
```

This example measures the total number of bytes per subscriber and protocol category (defined by a classifier) and resets the statistics each publish interval:

```
classifier "Category" values "Web" "P2P" "Other"
PolicyGroup {
  if expr(Flow.SessionProtocol = Protocol.Http) then \
    set Flow.Classifier.Category = Classifier.Category.Web
  if expr(Flow.IsApplicationProtocol("ares","bittorrent","edonkey","gnutella") then \
    set Flow.Classifier.Category = Classifier.Category.P2P
  if true then \
    set Flow.Classifier.Category = Classifier.Category.Other
}
measurement "BytesPerSubPerCategory" sum(Flow.Tx.ProtocolBytes) over publish_interval \
  unique by (Subscriber, Flow.Classifier.Category)
```

3.3.1.7 Top Measurement

Top measurements can be taken using an existing unique-by generic or sum measurement, or by using a statistical method to estimate top instances.

To identify the top instances of an existing unique-by generic or sum measurement, the syntax is:

```
measurement "<Name>" top(<n>, <measurement_reference>)
measurement "<Name>" topPercent(<n>, <measurement_reference>)
```

Where:

Name Name of the measurement.

top <n>	The number of top instances to identify as an absolute number. Must be a value between 1 and 1000.
topPercent <n>	The number of top instances to identify, as a percentage of the total number of instances. Must be a value between 1 and 100. If the percentage multiplied by the number of instances in the measurement being referenced is greater than 1000, only the top 1000 instances are identified.
<measurement_reference>	A reference to the value of a unique-by generic, sum, or other top measurement (`Measurement.Name`). The top n instances of this measurement are identified. Note that unique by subscriber measurements cannot be referenced in top measurements.

On a PTS 24000, 22000, or 14000, each individual module computes its own top instances and the final list of top instances is computed by compiling the lists from each module. In some cases, this can result in slight inaccuracies or the erroneous omission of instances that should have been reported near the bottom of the list. To reduce this error in reporting, the value of n should be set to a number larger than the number of instances you are interested in collecting. For example, if you are interested in the top 100 instances, configure your measurement to track the top 200 instances, then only look at the top 100 instances from that list.

If the number of unique instances defined by the unique_by_spec is too high to count with a generic or sum measurement, use this syntax:

```
measurement "<Name>" top(<n>, sum(<expression>)) over <period> {where <condition(s)>} unique
by <unique_by_spec>
```

The syntax is similar to that of the sum measurement, with the addition of the top operator.

The statistical method used to estimate the top instances will introduce some error into the measured values. Each instance that is measured by the top measurement will be no less than 95% of its actual value. It is also possible that some instances near the tail of the list will incorrectly be reported as being one of the top instances. This uncertainty is relatively higher when the volume of traffic being measured is low.

Example:
This example identifies the top 100 HTTP hostnames by connections:

```
measurement "TopHttp" top(100, sum(1)) over publish_interval \
  unique by (Flow.Client.Stream.Http.Host)
```

This example demonstrates how to publish a top measurement and count the top URLs by bytes.

```
table "HttpFlowState" (string "Url") timeout none unique by (Flow)
classifier "Url_Bytes" type string
PolicyGroup expr(Flow.SessionProtocol=Protocol.Http) {
  if expr(Flow.Client.Stream.Http.Host is not null) then \
    set Classifier.Url_Bytes = Flow.Client.Stream.Http.Url and \
    set Table.HttpFlowState.Url = Flow.Client.Stream.Http.Url
  if expr(Table.HttpFlowState.Url is not null) then \
    set Classifier.Url_Bytes = Table.HttpFlowState.Url
}

measurement "TopBytesByUrl" \
  top(6, sum(Flow.Tx.ProtocolBytes)) \
  over publish_interval \
  where expr(Flow.SessionProtocol = Protocol.Http) \
  unique by (Classifier.Url_Bytes)
publish "TopBytesByUrl" Measurement.TopBytesByUrl
if expr(Flow.SessionProtocol=Protocol.Http and Flow.IsEnd) then delete_row
Table.HttpFlowState
```

3.3.1.8 Expression Measurement

Expression measurements measure the value of a SandScript expression

The syntax is:

```
measurement "<Name>" expr(<expression>) {over <period>}
```

Expression measurements can only be used with scalar fields. This type of measurement is most often used to perform some math on two or more other measurements. For example, two bitrate measurements that measure upstream and downstream bandwidth can be added together to measure total bandwidth:

```
measurement "Upstream" bitrate where source class "internal"
measurement "Downstream" bitrate where destination class "internal"
measurement "Total" expr(Measurement.Upstream + Measurement.Downstream)
```

In this example, average packet size is determined by dividing total bytes by total packets:

```
measurement "Bytes" sum(Flow.Tx.Bytes)
measurement "Packets" sum(Flow.Tx.Packets)
measurement "AvgPacketSize" expr(Measurement.Bytes / Measurement.Packets)
```

3.3.1.9 Histogram Measurement

Histogram measurements measure floating point values of a SandScript expression.

Syntax:

```
measurement "Name" histogram(<histogram_reference>, <expression>)
 {over <period>} {where <condition(s)>} {unique by <unique-by_spec>}
```

Where:

histogram_reference — A SandScript reference to a histogram that defines the ranges of the bins that measured values are counted into (Histogram.Name).

expression — An integer or floating point valued expression to measure. Integer values are cast to floating point.

Alternately, you can perform the measurement using the increment action and by specifying which histogram bin to increment. For example:

```
measurement "MyHistogramMeasurement" histogram(Histogram.MyHistogram)over publish_interval
if <condition> then increment Measurement.MyHistogramMeasurement bin <expression>
```

You can use the `by` parameter to the increment action to increment a histogram by more than 1. For example:

```
if <condition> then increment Measurement.MyHistogramMeasurement bin <expression> by 2
```

3.3.2 Measurement Groups

Measurement groups can be used to optimize policy by grouping measurements with similar conditions together, or to create a decision tree.

They may contain measurement definitions or nested measurement groups. The syntax and behavior is the same as a policy group except measurements instead of rules are declared within the group.

- The condition on the measurement group controls entry into the group. If the condition matches, the measurements are performed, otherwise they are not.
- If the keyword "all" is specified, all of the measurements in the measurement group that match their condition are performed. If "all" is not specified, only the first measurement that matches its condition is performed.
- Measurement groups can be nested.
- Measurement groups have an if-elseif-else syntax.

Example:

```
MeasurementGroup {
 measurement "GnutellaConnections" connections where \
        expr(Flow.ApplicationProtocol = Protocol.Gnutella)
 measurement "HttpConnections" connections where  \
        expr(Flow.ApplicationProtocol = Protocol.HTTP)
 measurement "BittorrentConnections" connections where \
         expr(Flow.ApplicationProtocol = Protocol.BitTorrent)
 measurement "EdonkeyConnections" connections where \
        expr(Flow.ApplicationProtocol = Protocol.Edonkey)
}
MeasurementGroup  transfer uni and sender class "internal" {
 measurement "GnutellaUploads" connections where \
        expr(Flow.ApplicationProtocol = Protocol.Gnutella)
 measurement "HttpUploads" connections where \
        expr(Flow.ApplicationProtocol = Protocol.HTTP)
 measurement "Bittorrent Uploads" connections where \
        expr(Flow.ApplicationProtocol = Protocol.BitTorrent)
 measurement "Edonkey Uploads" connections where \
        expr(Flow.ApplicationProtocol = Protocol.Edonkey)
}
```

Example:

```
# measure HTTP connections and bytes
# this policy is optimized by hoisting the common protocol condition
# up into a measurement group
MeasurementGroup expr(Flow.ApplicationProtocol = Protocol.HTTP) all {
 measurement "HttpConnections" connections
 measurement "HttpBytes" sum(Flow.Tx.ProtocolBytes)
}
# use a different measurement to measure new connections per tier
# declare them inside of a measurement group since the conditions
# are mutually exclusive (avoids needlessly evaluating all conditions
# stop as soon as you find the one that matches
attribute "tier" values "gold" "silver" "bronze"
MeasurementGroup expr(Flow.ApplicationProtocol.IsNew) {
 measurement "GoldNewConnections" sum(1) subscriber "tier"="gold"
 measurement "SilverNewConnections" sum(1) subscriber "tier"="silver"
 measurement "BronzeNewConnections" sum(1) subscriber "tier"="bronze"
}
```

3.3.3 Histograms

Histograms are declared through policy to define bins for use with histogram measurements and controllers.

Histogram bins cover a continuous range of floating point numbers from -infinity to +infinity. A single histogram bin is represented by a lower and upper bound value, where the lower bound is exclusive, and the upper bound is inclusive.

Histogram types define how the bins are distributed over the range of floating point numbers. All histogram types have implied lower and upper outlier bins. The maximum number of bins is 200, not including the outlier bins.

The supported histogram types are:

- Linear - Evenly spaced bins defined over a range of floating point numbers.
- Custom - Consists of explicitly defined bin ranges.

Syntax:

```
histogram "<name>" <type> <parameters>
```

Where:

<name> The name of the histogram. The name is used to identify the histogram through CLI and SandScript.

- Must begin with a letter and can only contain letters, digits, and underscores.
- Must not be a reserved SandScript keyword.
- Is not case sensitive.

<type> Specifies the histogram type. Refer to the details about specific histogram types that follow.

<parameters> Type-specific parameters for the histogram.

3.3.3.1 Linear Histogram

Linear histograms are defined by a minimum value, maximum value, and number of bins.

Syntax:

```
histogram "<name>" linear min <min_val> max <max_val> bins <num_bins>
```

Where:

- min_val is the lower bound of the first bin
- max_val is the upper bound of the final bin
- num_bins is the number of bins in the histogram.

Bins are of width (max - min)/bins, with all of the bins covering the range {min, max). The lower and upper outlier bins are defined over the ranges {-infinity, min), and {max, +infinity} respectively.

3.3.3.2 Custom Histograms

Custom histograms are defined by a consecutive list of floating point values. The number of bins is one less than the number of values given (plus the two implied outlier bins).

Syntax

```
histogram "<name>" custom (<value1>, ... , <valueN>)
```

Where: value1..valueN are consecutive floating point values indicating the bin ranges for the histogram.

The range of each individual bin is defined by two consecutive values in the list, where the first value is the lower bound and the second value is the upper bound of the bin. The lower and upper outlier bins are defined over the ranges {-infinity, first), and {last, +infinity}, where first and last are the first and last values in the list of floating point values.

Example:
This example defines a linear set of histogram bins that divide the range 0-100 into 100 bins of the same size.

```
histogram "LinearHistogram" linear min 0 max 100 bins 100
```

Example:
This example defines a custom set of histogram bins from 0-100 with finer granularity over numbers closer to zero.

```
histogram "CustomHistogram" custom (0, 0.5, 1, 1.5, 2, 3, 5, 10, 20, 30, 50, 70, 100)
```

Example:
Using the histogram distributions defined above, this example keeps a running count of the entire network's round-trip time (RTT) measured into a custom histogram.

```
measurement "MeasureRttIntoCustom" histogram(\
     Histogram.CustomHistogram, \
 ToMilliseconds(Flow.Subscriber.HandshakeRTT)) \
 where expr(Flow.Subscriber.HandshakeRTT is not null)
```

Example:

Using the histogram distributions defined above, measure and publish the RTT per protocol into a linearly distributed histogram:

```
measurement "MeasureRttIntoLinear" histogram(\
     Histogram.LinearHistogram, \
 ToMilliseconds(Flow.Subscriber.HandshakeRTT)) \
 over publish_interval \
 where expr(Flow.Subscriber.HandshakeRTT is not null) \
 unique by (Flow.ApplicationProtocol.StatsProtocol)
publish "RttByProtocol" measurement.MeasureRttIntoLinear
```

Example:
Measure and publish a histogram of the number of bytes per flow unique by protocol. Use a histogram distribution with bins of size 500 bytes over the range 0-100000.

```
histogram "Bytes" linear min 0 max 100000 bins 200
measurement "FlowBytesPerProtocol" \
    histogram(histogram.Bytes, Flow.Rx.TotalBytes + Flow.Tx.TotalBytes) \
    over publish_interval \
    where expr(Flow.IsEnd) \
    unique by (Flow.ApplicationProtocol.StatsProtocol)
publish "FlowBytesPerProtocol" measurement.FlowBytesPerProtocol
```

Example:
Measure flow duration in seconds into a custom histogram. The bins are distributed such that flow durations of a few seconds up to an entire day can be measured.

```
histogram "Seconds" custom(0, 1, 2, 3, 4, 5, 10, 20, 30, 60, 300, 600, 1800, 3600, 21600,
 43200, 86400)
measurement "FlowAge" histogram(histogram.Seconds, Flow.Age) \
    over publish_interval \
    where expr(Flow.IsEnd)
publish "FlowDuration" measurement.FlowAge
```

3.4 Published Expressions

Published expressions facilitate custom reporting and provide a mechanism for reporting on the value of a SandScript expression over time.

The value of the SandScript expression is periodically sampled and stored in the database. These values can then be reported on via Network Demographics Server reports. To examine published expressions, create a Published Expressions report (in Network Demographics, choose **Network Characterization** > **Demographics** > **by Network Element** > **Published Expressions**).

The sampling period defaults to 15 minutes (900 seconds), but can be configured using the `set config service statistics log-interval published-expression <value>` CLI configuration command. Note that the sampling period is a global variable which affects all published expressions.

The SandScript expression being published must be a scalar, integer-valued expression. A scalar expression is one whose value is not tied to a particular flow, packet, or subscriber.

```
publish "<name>" <expression> {aggregate sum|max}
```

<name> The name assigned to the expression. The name can be any alphanumeric string, unique among publish statements. This name appears in Network Demographics reports.

<expression> Any scalar, integer-valued expression.

aggregate {sum|max} The aggregation type of the published expression. This option must be specified if the expression being published does not have an aggregation type. Non-generic measurements have an implied aggregation type and therefore do not need this option. The aggregation types are:

- sum - The published expression is treated as a counter when its values are aggregated across modules and over time intervals in reports.
- max - The published expression is treated as a gauge where the peak value of the published expression is taken, across modules and over time intervals in reports.

on-scalar measurements can be published if they have a particular unique by specification and are either synchronized to the publishing interval or over infinity. For a non-scalar measurement to be publishable, it must be unique by one of:

- Subscriber.
- Subscriber and a single classifier.
- Subscriber and Flow.ApplicationType.
- Flow.Application.
- Flow.Application and 1-3 classifiers.
- Flow.ApplicationType.
- Flow.ApplicationType and 1-3 classifiers.
- 1-4 classifiers.
- Subscriber.
- Subscriber and classifier.
- 1-4 classifiers.

When publishing unique by subscriber or subscriber and classifier, the number of published instances for any given subscriber is limited by the capacity of the SPB. Counters are defined as measurements which are aggregated using the sum operator and gauges are defined as measurements which are aggregated using the max operator. The limits are:

SPB Release	Counter Limit	Gauge Limit
6.20 and later	64	12
6.00 and earlier	32	6

Example:
For example:

```
classifier "A" type string {
    if true then set Flow.Classifier.A = "A"
}

classifier "B" type string {
    if true then set Flow.Classifier.B = "B"
}

classifier "C" type string {
    if true then set Flow.Classifier.C = "C"
}

measurement "BytesPerProtocolABC" \
  sum(Flow.Rx.Bytes) over publish_interval \
  unique by (Flow.ApplicationProtocol.StatsProtocol, \
  Flow.Classifier.A, Flow.Classifier.B, Flow.Classifier.C)

publish "BytesPerProtocolABC" Measurement.BytesPerProtocolABC
```

This example declares a PTS measurement to measure BitTorrent upload bandwidth, and then publishes the value of that measurement.

```
measurement "BittorrentUp" connections where \
expr(Flow.ApplicationProtocol = Protocol.BitTorrent) \
```

```
and transfer uni and receiver class "external"
publish “BittorrentUploads” measurement.BittorrentUp
```

This example configures session management of Gnutella uploads, and then publishes the number of TCP resets sent out by the PTS.

```
limiter "GnutellaUploads" 10 connections where \
expr(Flow.ApplicationProtocol = Protocol.Gnutella) and \
receiver class "external" and expr(not(Flow.IsReset))
if expr(limter.GnutellaUploads.limit) then tcp-reset
publish “GnutellaResets” limiter.GnutellaUploads.LimitedFlows
```

This example shapes HTTP traffic to 20 Mbps. The effectiveness of the shaper is demonstrated by publishing the bandwidth into the shaper, and the bandwidth out of the shaper.

```
shaper "HTTPShaper" 20Mbps 15Kbytes
if protocol "http" then shape to subscriber shaper "HTTPShaper"
publish “ShaperRxRate” Shaper.HTTPShaper.CurrentRateIn
publish “ShaperTxRate” Shaper.HTTPShaper.CurrentRateOut
```

3.5 Timers

Timers are used to generate an event at some point in the future. When the event occurs, SandScript rules available in the timer's context are evaluated.

There are two types of timers: repeating and non-repeating. Repeating timers will be immediately rearmed every time they are fired. Every time a repeating timer is armed, it will use the same duration. The duration of repeating timers cannot be changed. Non-repeating timers will only fire once, unless they are rearmed by the timer arm action.

Information about timers can be viewed by using the `show policy timers` CLI commands. Timers can be explicitly armed and cleared in SandScript using timer arm and timer clear actions.

```
timer “<Name>” {arm <timer_interval> {repeat true|false}} {tolerance <maximum_lag>|infinite}
{resolution <accuracy>}
```

Where:

Name

Name of the timer, used to identify the timer in the CLI and when referencing the timer using SandScript expressions. Note that it:

- Must begin with a letter and can only contain letters, digits, and underscores.
- Is not case sensitive.
- Must not be a reserved SandScript keyword.
- Must be unique amongst timers.
- Determines the timer type (`Timer.Name`) that can be used as column types of a SandScript table.

arm <timer_interval> {repeat {true|false}}

Sets the interval that the timer should be armed for as soon as it is created. If omitted, the timer will be disarmed by default and will not be repeating.

<timer_interval> is an integer followed by units of time, for example, 5 seconds, 500 milliseconds.

The optional repeat clause specifies whether or not the timer is repeating. If omitted, the default value is false.

tolerance {<maximum_lag>|infinite}

Indicates the maximum amount of time that a timer can fire late before it is considered to be an error. If this tolerance is exceeded, an alarm is raised.

<maximum_lag> is an integer followed by units of time, for example, 5 days, 5 hours, 5 minutes, 5 seconds, or 500 milliseconds. The default value is infinite, which indicates that it is never considered an error regardless of how late the timer fires.

resolution <accuracy> The accuracy requirement of the timer. A timer with a duration of 10 seconds will be expected to fire no later than (10 seconds + accuracy) after it is armed. For example, a timer with a duration of 10 seconds and a resolution of 10 milliseconds will be expected to fire 10 to 10.01 seconds after it is armed. <accuracy> is an integer followed by units of time, for example, 5 seconds, 500 milliseconds. The default value is 100 milliseconds.

Example:
For example:

```
# Define a timer that fires every minute with one second resolution
timer "MinuteTimer" arm 1 minutes repeat true resolution 1 seconds
PolicyGroup expr(Timer.MinuteTimer.Expired) all
{
  # Rules to evaluate every minute when the timer expires
}
```

3.5.1 Timer Arm

Use the timer arm action to arm a policy timer for a particular duration.

```
timer.<TimerName>.arm({duration:<time>})
```

Where:

<TimerName> is the name of the timer

<time> indicates how long to arm the non-repeating timer for. If the duration argument is specified, arms non-repeating timer for the duration time. If the duration argument is omitted, the timer is reset to its initial duration. A negative duration is treated the same as a duration of zero. (expires now).

3.5.2 Timer Clear

Use the timer clear action to disarm a policy timer. This action may be used on both repeating and non-repeating timers. After a timer has been disarmed, it may be restarted using its arm action.

```
timer.<TimerName>.clear()
```

3.6 Table Syntax

Tables are used to store state and build state machines in policy.

Tables in policy are similar to tables in a database. Tables have a name and a set of named columns. Each table must define at least one column. Default values can be assigned to columns. Rows are created when a column value is set in policy using a set action, or an increment action, or the Create function is used. Rows are deleted either explicitly using the delete_row action or when they time out if configured to do so.

Warning:
Any change to the table schema deletes all rows from the table upon reload.

When the definition of a table in policy changes, for example because new columns are added, data types are changed, or you change the generation number, any existing data in the table is cleared when these changes are loaded. If you do not wish to lose the data, instead of deleting a column, just ignore it. Or, if you wish to add a column, try adding new tables to achieve the functionality that is desired without changing existing tables.

By default, a maximum of 1000 tables can be defined in policy.

There is a maximum amount of memory that can be used by table rows and a limit on the number of rows that can be created. An alarm is raised if either limit is exceeded, and attempts to create new rows will fail until the active usage falls below the maximum.

You can view information about tables by executing the CLI command `show policy tables`.

```
table "<Name>" (<type> "<ColumnName1>" {default <expression>}{sparse true|false},
    <type> "<ColumnName2>" {default <expression>}{sparse true|false}, …)
timeout <interval> {reset_timeout read|write|readwrite|none } {index "SecondaryIndexName"
(PAL_expression)} {generation <number>}
unique by <unique_by_spec>
```

<Name> Name of the table, used to identify the table in the CLI and when referencing the table using SandScript.

- Must begin with a letter and can only contain letters, digits, and underscores.
- Is not case sensitive.
- Must be unique amongst tables.
- Determines the name of the SandScript fields (Table.Name) that are used to access rows in the table.

<type> Type of the column. Supported types include: string, string(N) (string of length at most N), float, integer (use smaller integer types to use less memory: integer8, integer16, integer32, unsigned8, unsigned16, unsigned32), timestamp, ipaddress, boolean, and `Timer.TimerName`.

<ColumnName> Name of a column. Multiple columns can be specified. Column specifications (which include the name, the type, and column options) are separated by commas. The same rules that apply to the table name apply to the column names. The column names (along with the table name) determine the name of the SandScript fields that are used to read and write to the table (for example, `Table.Name.ColumnName`).

default <expression> The default value of the column when a row is created. Optional, if omitted, null is assumed. The expression must be assignment compatible with the type of the column.

sparse {true|false} Specifies if the table will optimize the memory used by the column under the assumption that the column usually holds a null value. This option should only be set to true for columns that are null most of the time or for timers that are unarmed most of the time. This option cannot be used for key columns or timer columns that are armed by default. Sparse is optional; if omitted the column is not sparse.

timeout <interval> Specifies how long after a row is created should it time out, after which it is deleted. <interval> is an integer followed by units of time. Examples are:

```
5 seconds
5 sec
500 milliseconds
500 ms
```

Use “none” to indicate that rows should never time out. Specifying a 0-length time interval has the same effect as using “none”. If using none, rows must be deleted using the delete_row action. Take care to ensure rows are deleted in a sufficient time such that resources are not exhausted.

reset_timeout read|write|readwrite|none Specifies when the timeout time of a row is reset. If omitted, the default value is readwrite.

- Read causes the timer to reset each time a value is read from the row.
- Write causes the timer to reset each time a value is written to the row.
- Readwrite causes the timer to reset each time a value is read from or written to the table.
- None causes the timer to never reset, the timer will start when the row is created and the row will be removed when it expires.

index "SecondaryIndexName" (PAL_expression)	Defines a secondary index in table. This provides access to all records with the same column value via the foreach statement. PAL_expression must refer to one of the columns of the table on which the index is being defined. You cannot define multiple indexes on different columns. For example: `"index "MyIndex" (Table.MyTable.MyColumn)"` An index cannot have the same name as a table column. You cannot index timer and boolean columns.
generation <number>	A non-negative integer that specifies if a table should be cleared on policy reload. If the generation number changes, the table will be cleared upon reload.
<unique_by_spec>	The key for the table, a separate row is created for each instance defined by the specification. It is defined as a SandScript expression list (a comma separated list of fields or expressions inside parenthesis), for example, “unique by (Flow)”. Note that columns of the table may be used in the unique by specification, accessed via their SandScript field, essentially creating a primary key for the table, for example, “unique by (Table.Name.ColumnName)”.

Example:
For example:

```
# define the table
# this table can store a string per Flow
table “FlowState” (string “HttpHostname”) timeout none unique by (Flow)
PolicyGroup expr(Flow.SessionProtocol = Protocol.Http) {
  # when HTTP flows end, delete their row from the table
  if expr(Flow.IsEnd) then delete_row Table.FlowState
  # for every HTTP flow, create a row and store the hostname
  if expr(Flow.Client.Stream.Http.Host is not null) then \
    set Table.FlowState.HttpHostname = \
      Flow.Client.Stream.Http.Host
   }
```

3.6.1 Accessing Table Rows

Table rows can be accessed in one of two ways, either implicitly (the expressions in the unique_by_spec specify a particular row when evaluated) or explicitly using square bracket “[ ]“ accessors.

For implicit access, tables are accessed via these SandScript fields:

```
Table.Name.ColumnName
```

For explicit access, tables are accessed via these SandScript fields:

```
Table.Name[<Key>].ColumnName
```

Note that if columns of the table are used in the unique_by_spec, only explicit access is supported. This is not strictly true if the table has a timer column, which defines a new timer context in which a row is accessed implicitly. For example:

```
timer “MyTimer”
table “MyTable” (integer “IntCol”, \
                 string “StringCol”, \
                 Timer.MyTimer “TimerCol”) \
    timeout none unique by (Table.MyTable.IntCol)

PolicyGroup expr(Table.MyTable.TimerCol.Expired) {
    # Table.MyTable.IntCol and Table.MyTable.StringCol accessed implicitly in this timer expired
 context
}
```

Use the delete_row action with the `Table.Name` or `Table.Name[<Key>]` fields to delete rows from a table.

3.6.2 Table Clear

Use the table clear action to delete all rows in a SandScript table.

```
table.<TableName>.clear()
```

3.6.3 Table Decrement

Use the table decrement action to efficiently decrement multiple integer columns of a row in a SandScript table.

An attempt to increment a read-only column results in an error. Supported for integer columns only. If the row for that key is not in the table, a new row is created.

```
table.<TableName>[<Key>].decrement(ColumnName1:<expression>, ColumnName2:<expression>, ...)
```

Where <expression> is the amount by which to decrement the column.

3.6.4 Table Increment

Use the table increment action to efficiently increment multiple integer columns of a row in a SandScript table.

An attempt to increment a read-only column results in an error. Supported for integer columns only. If the row for that key is not in the table, a new row is created.

```
table.<TableName>[<Key>].increment(<ColumnName1>:<expression>, <ColumnName2>:<expression>, ...)
```

Where <expression> is the amount by which to increment the column.

3.6.5 Table Reset

Use the table reset action to efficiently reset all writable columns of a row in a policy table to their default values.

```
table.<TableName>[<Key>].reset()
```

3.6.6 Table Set

Use the table set action to efficiently set multiple columns of a row in a SandScript table.

An attempt to set a read-only column results in an error. The order in which columns are set is not guaranteed. If order is important, multiple set actions should be used instead. If the row for that key is not in the table, a new row is created.

```
table.<TableName>[<Key>].set(<ColumnName1>:<expression>, <ColumnName2>:<expression>, ...)
```

Where <expression> is the value to set for the column.

3.6.7 Table Timer Arm

Use the table timer arm action to arm a timer column of a row in a policy table. The action only applies if <ColumnName> is a timer column, and the timer is only armed for the indexed row. Refer to the Timer arm action.

```
table.<TableName>[<Key>].<ColumnName>.arm({duration:time})
```

3.6.8 Table Timer Clear

Use the table timer clear action to disarm a timer column of a row in a policy table. The action only applies if <ColumnName> is a timer column, and the timer is only disarmed for the indexed row. Refer to the Timer clear action.

```
table.<TableName>[<Key>].<ColumnName>.clear()
```

3.7 User-defined Functions

Functions can be defined to group together commonly-executed rules, perhaps parameterized by a set of argument values.

Effective use of functions can assist with writing cleaner, more modular, and more maintainable SandScript policy. Functions can be invoked after they are defined, as either expressions or actions. A function cannot call itself or a function that is defined after it. If a function does not define a return value, then it returns null when used as an expression. The function body, however, is still executed.

```
function "<name>" ({writable} <type> "<ArgName1>" {default <expression>},
                   {writable} <type> "<ArgName1>" {default <expression>},...)
                   {returns <type> "<returnValName>"}
{
         rules and/or nested SandScript groups...
}
```

<name> Name used to invoke the function. Note that name:

- Must begin with a letter and can only contain letters, digits, and underscores.
- Must not be a SandScript reserved keyword.
- Is not case-sensitive.
- Cannot conflict with other function names (user-defined and internal).

writable Allows the function to modify the expression assigned to the argument when the function is invoked. The writable keyword may not be used if the default keyword is used.

default Specifies a default value to use for the argument if the argument is not specified when the function is invoked. The default keyword may not be used if the writable keyword is used.

<type> Type of argument or return value. Supported types include all those supported for local variables.

<returnValName> The return value name which can be set from within the function body. If the field is not set within the function body, the return value will be null.

Example:
This example invokes the user-defined function FuncName:

```
if <condition(s)> then FuncName(ArgName:<arg_expression>, ArgName:<arg_expression>, ...)
if expr(FuncName(ArgName:<arg_expression>, ArgName:<arg_expression>, ...) then action(s)
if <condition(s)> then set field = FuncName(ArgName:<arg_expression>,
ArgName:<arg_expression>, ...)
```

Where:

- FuncName is the name of the function as provided in the definition.
- ArgName is the name of a function argument from the definition.
- <arg_expression> is the expression to be assigned to the corresponding ArgName argument when the function body is executed. The expression must be assignment compatible with the type of the argument. If the corresponding ArgName argument is writable, then the expression must be writable and of the same type as the argument.

Example:
A simple summation example:

```
function “add” (integer “LeftHandSide”, integer “RightHandSide”) returns integer “total”
{
    if true then set total = LeftHandSide + RightHandSide
}


PolicyGroup all {

        integer "totalBytes"
        if true then set totalBytes = add(LeftHandSide:1,RightHandSide:2)

}
```

3.8 Foreach

Foreach loops can be used to efficiently evaluate rules for every row in a policy table.

A foreach loop is useful when it is necessary to extract information from or update information in many rows of a policy table, as it is more efficient than accessing rows individually. The foreach policy construct can be used with Diameter policy.

```
foreach “<IteratorName>” in Table.<TableName>{
    rules and/or nested policy groups...
}
```

<IteratorName> The name of the foreach iterator, which is of type Table.TableName:cursor. The iterator has scope only within the foreach construct and cannot be explicitly re-assigned.

- Must begin with a letter and can only contain letters, digits and underscores.
- Must not be a policy reserved keyword.
- Is not case sensitive.
- Cannot conflict with other local variables that are still in scope.

<TableName> The name of the policy table being accessed.

Example:
For example:

```
table "MyTable" (integer "key", integer "value") timeout none \
    unique by (Table.MyTable.Key)
foreach "row" in Table.MyTable
{
    # For all rows with even-numbered keys, increment the value
    if expr((row.key % 2) = 0) then increment row.value
}
```

3.9 Events

Events can be defined in SandScript to allow one application to raise an event that other applications can listen for and take action on.

When SandScript raises an event, all of the event's arguments are evaluated, and those values are available to any rules using fields of the event. All event arguments are optional. If they are not provided when the event is raised, event listeners observe a null value for those arguments.

There is no guarantee on when an event will be run, or on the relative ordering of events. There may be any number of rules using the event, or none. The `Event.Name` SandScript field is only evaluated when the event has been raised. Actions within an event listener cannot raise the same event that is being listened for, or an event that will cause the same event to be raised. Events cannot be raised in rules that are available in all SandScript scopes.

Events may be redefined as long as the types and names of all event arguments are the same in each definition. The order of the event argument definitions does not matter.

```
event "<Name>" (<type> "<ArgName1>", <type> "<ArgName2>", ...)
```

Where:

- <Name> is the name of the event. Note that <Name>:
  - Must begin with a letter and can only contain letters, digits, and underscores.
  - Must not be a SandScript reserved keyword.
  - Is not case sensitive.
- <type> is the type of the event argument. Supported types include: string, float, integer, boolean ipaddress and Record.RecordName.

Events are raised as actions as:

```
if <condition(s)> then Event.<Name>(ArgName:<arg_expression>, ArgName:<arg_expression>, ...)
```

Where:

<Name>	The name of the event as provided in the definition.
<ArgName>	The name of an event argument from the definition
<arg_expression>	Is assigned to the corresponding ArgName argument when the event is raised. The expression must be assignment compatible with the type of the argument.

SandScript can listen for an event. For example:

```
if expr(Event.Name) then action(s)
```

Where: <Name> is the name of the event as provided in the definition.

Example:
This example raises an event on every new flow. When the event is run, a row is inserted in the table. If the flow was an HTTP flow, the string HTTP is stored in the table, otherwise, NULL is stored in the table. The remainder of the SandScript evaluation happens before the event is executed. When the SandScript evaluation completes, SandScript is run for all events in the order in which they were raised. All events will execute before another non-event SandScript evaluation occurs.

```
table "ExampleTable" (integer "key", string "value") timeout none unique by
(Table.ExampleTable.Key)
event "ExampleEvent" (integer "IntegerArg", string "StringArg")

PolicyGroup expr(Flow.IsNew) {
   if expr(Flow.SessionProtocol = Protocol.Http) then Event.ExampleEvent(IntegerArg: Flow,
 StringArg: "HTTP")
    if true then Event.ExampleEvent(IntegerArg: Flow)
}

if expr(Event.ExampleEvent) then set ExampleTable[Event.ExampleEvent.IntegerArg].Value =
Event.ExampleEvent.StringArg

if expr(Flow.IsEnd) then delete_row Table.ExampleTable[Flow]
```

3.10 Protocol Definitions

Protocol matching definitions can be written in the policy configuration file so they can be used to write policy actions.

```
protocol {user<number> re TCP|UDP {!} init|resp "<regular_expression>" {priority
<priority_number>}}
```

user<number> The user-defined protocol to use. Use if a library is not yet available for a specific protocol or if there is a requirement to customize a protocol. Number can be in the range 1-32. For example user1.

TCP|UDP Match either TCP or UDP flows. If not specified, the match defaults to TCP flows.

! The `not` regular expression condition. Use when you want the protocol match to fail.

init|resp Indicates if the regular expression should be treated as an initiator (init) or a responder (resp) which is the default. If neither is specified, the regular expression is treated as both.

“<regular_expression>” A regular expression compliant to IEEE Std. 1003.2 (POSIX.2) with a maximum length of 192 characters. Examples of regular expressions are:

- “XYZ” - XYZ anywhere in approximately the first 1000 bytes of the first packet
- “\AXYZ” - XYZ at the start of the session. \A is anchor.
- “WX.Z” - WX followed by any character followed by Z. The “.” is a wildcard character.
- “\A{0-9}*” - any number 0 to 9 repeated any number of times (including zero times) at the start of a packet.

A regular expression can also be specified in hexadecimal. Hexadecimal characters are specified as \x## where \x is the hexadecimal prefix and ## specifies the hexadecimal value, for example \x20. To match a UDP protocol with the following byte sequence:

```
protocol user1 re UDP \
"\`\x11\x22\x33\x44\x55\x66\x77\x88\x99\xaa\x11\x22\x33\x44\x55\x66" \
priority 100
```

priority <priority_number> Priority for the protocol. All protocols have an associated priority so that a regular expression can be invoked either before or after other protocols. The <priority_number> can be from 0 to 65535, going from the highest priority (0) to the lowest priority (65535). If no priority is specified, the protocol will default to the standard internal priority for all protocols which is 100.

3.10.1 Disabling Matching for a Protocol

Support for a built-in protocol can be disabled.

This is useful since the detection of some protocols is more CPU intensive than others. Note that disabling matching for a protocol results in the protocol being unclassified. To no longer inspect (or) match traffic for a protocol, mark it as disabled in the policy.conf file. For example:

```
protocol "winmx" disable
```

Or, you can disable all protocol recognition using these steps.

Note:
Port matching is not affected by disabling all protocols. If you wish to disable port matching as well, it must be done separately.

1. Log in to the PTS as an administrative user, then start the CLI using the command: `svcli`.

   You can also access the CLI using the command `cli`.

2. At the CLI prompt run these commands:

```
PTS> configure
PTS# set config service protocol tcp enabled false
PTS# set config service protocol udp enabled false
```

3. After you have completed your configuration on the central PTS, commit the changes and restart (which may impact service). Run: `PTS# commit`

4

Subscriber to IP Mapping

4.1 Subscriber Aware SandScript Policy

Subscriber awareness is achieved through an IP address to subscriber mapping, and requires an SPB to be configured and connected to the PTS or SDE.

The ability to map a subscriber is a pre-requisite for any subscriber-aware policies, such as subscriber statistics collection or Top Talkers. An active IP address and optional site number to subscriber mapping is called a session for that subscriber. To identify a session, use the field `Session.Id`. This field returns an integer representing a unique identifier for this subscriber session or NULL if no subscriber session exists.

Attributes are type-value pairs, such as “servicelevel”=”gold”, that can be attached to a particular subscriber or to any of that subscriber's sessions. Each subscriber, or session, may have any number of attributes. These attributes may be used in conditions and typically represent the type of SandScript and type of service the subscriber will receive (for example, collect statistics, shape traffic, allow/deny and so on).

The PTS maintains subscriber sessions as long as the subscriber has active traffic flowing through the element. Once the subscriber becomes inactive, the mapping is timed out after a configurable timeout period.

The SDE maintains a subscriber session as long as SDE SandScript continues to use the session.

4.1.1 Updating SandScript to Uniquely Identify Subscribers

You must update SandScript policy that uses a subscriber's IP address to uniquely identify a session to use the Session.ID field instead.

A common use-case in legacy custom SandScript is to use a subscriber's IP address as a unique handle for a subscriber session, for example in unique-by measurements, or as a table key. In cases where subscriber IP addresses overlap, a subscriber's IP address is no longer unique. Therefore, these policies must be updated to use the field Session.ID to uniquely identify a subscriber's session. The Session.ID field returns an integer identifier that is unique per session.

It is safe to use this field even if session qualifiers are not enabled, so Sandvine recommends updating any policies that use unique-by IP address to use Session.ID instead.

4.1.2 Subscriber Lookup

Subscriber to IP address mappings and all subscriber or session attributes are maintained by the SPB, which pushes notifications to, and accepts queries from, the PTS or SDE elements as needed. When traffic from an unknown subscriber IP address is received by the PTS or SDE, the subscriber who owns that IP address and all attributes of that subscriber are looked up.

To reduce network congestion, the PTS and SDE each use an algorithm which may apply a delay before either element's first look up request is sent to the SPB for any newly-discovered subscriber IP address. The algorithm operates one of these modes:

- Static mode (the default) has a pre-configured delay that does not change. If the static mode is enabled and the delay set to 0, there is no delay incurred on any lookup request sent by the PTS.
- Dynamic mode adjusts the size of delay based on the number of unnecessary lookups. An unnecessary lookup is one where a look up was requested but the IP becomes mapped by a push notification before the lookup response arrives. If there are too many unnecessary lookups, the delay is increased and when the percentage of unnecessary lookups drops, the delay is decreased.

These CLI commands can be used to manage subscriber lookup:

CLI commands	Description	Allowed	Default
show service subscriber-management stats	Displays the current delay time before the PTS will send a lookup request to the SPB.	N/A	N/A
set config service subscriber-management lookup initial-delay mode	Sets the lookup mode.	dynamic static	static
set config service subscriber-management lookup initial-delay static delay	If configured, bypasses dynamic lookup and sets a delay time for all lookups. Units of measure for configuration is milliseconds, therefore, the maximum is equivalent to 100 seconds.	0-100000	0

4.1.3 Populating Subscriber to IP Address Mappings

Subscriber-IP mapping can be populated dynamically using DHCP, RADIUS, or GTP-C, or statically using a script.

In situations where Remote Authentication Dial-In User Service (RADIUS) Accounting is deployed, the mapping can be obtained from the accounting records. Where DHCP is used, the mapping can be obtained by intercepting DHCP traffic. Sandvine employs a highly flexible, SandScript-based application called Subscriber Mapping that can be used in a wide variety of environments to populate subscriber to IP address mappings.

For detailed information on configuring these methods for subscriber-IP mapping, including instructions for the PopulateSubIpMap script, refer to the *Subscriber Mapping User Guide*.

4.2 Subscriber Database Configuration

The database query behavior and subscriber management strategies such as timing out subscribers may need to be configured, using the appropriate CLI configuration command.

Normally, subscriber mapping behavior should not need to be changed from the default. Any changes should be done under the direction of Sandvine Customer Support as inappropriate configuration can have a detrimental effect on element and database performance.

Subscriber Database Query Configuration

These CLI configuration commands are used to define the subscriber database query behavior:

CLI Configuration Command	Description	Allowed	Default
set config service spb version	Configures the SPB version that the PTS or SDE is connected to. Select the major version of the SPB when configuring. For example, choose “5.60” if connected to an SPB running any maintenance release of 5.60.	6.20, 6.00, 5.60, 5.51, 5.40	Newest supported SPB version
set config service spb servers	Contains a space-separated list of hostnames and/or IP addresses of SPB devices the PTS or SDE connects to. The PTS or SDE connects to only one of the servers in this list, but if the connection fails it will fallback to other servers. By default, the PTS or SDE uses an ssl-encrypted connection; if any server is configured with a tcp:// prefix, the PTS or SDE will use an un-encrypted TCP connection.	Any valid host name or IP address, with optional tcp:// prefix	Empty

Subscriber Management Configuration

These CLI configuration commands are used to define the subscriber management behavior.

CLI Configuration Command	Description	Allowed	Default
set config service subscriber-management timeout inactivity	Sets the timeout length (in seconds) after which a subscriber-IP mapping expires if no traffic is received for that IP address. When traffic is again seen for that IP, a lookup will be issued.	1 to 1209600 seconds (14 days)	1200
set config service subscriber-management timeout lookup initial	Sets the initial delay between subscriber IP lookup requests. The delay between requests doubles with each request.	1 - 86400 seconds (24 hours)	120
set config service subscriber-management timeout lookup max	Sets the maximum possible delay between subscriber IP lookup requests.	1 - 86400 (24 hours)	32400
set config service subscriber-management timeout late	Sets the maximum delay between the initial lookup request for a subscriber IP and that IP becoming mapped, after which the mapping is considered “late”. Related alarm is Alarm Model 30: Subscriber mapping on PTS is delayed.	1 - 120	30
set config service subscriber-management login-events handle-notification	Configures the PTS or SDE to process IP assignment notifications from the SPB	true false	true
set config service subscriber-management lookup max-attempts	Maximum number of unsuccessful subscriber-IP lookup attempts the PTS or SDE will do before giving up. Set to 0 for no limit.	0 - 255	0
set config service subscriber-management lookup on-receive	Configures the PTS or SDE to perform subscriber lookups when the subscriber IP receives unidirectional data. **Note:** The PTS or SDE always does subscriber lookups when the subscriber IP is sending data.	true false	true

4.3 Disabling Secure Tunnels Between the SPB and a PTS Element

By default, communication between the SPB and a PTS element is encrypted using a secure tunnel.

In situations where messaging performance is critical, such as when the rate of subscriber mappings is higher than 4000 mappings/sec, or the volume of statistics published is high, disabling security tunnels will improve messaging performance on the PTS.

To disable security tunnels, prefix the SPB element's IP address with `tcp://` when entering addresses in the Quickstart menu, or in the CLI configuration mode. For example, using `tcp://10.10.10.1` creates a connection bypassing the security tunnels.

To verify that the SPB is properly connected, at the CLI prompt enter `show service spb connections`. Sample output:

```
Connections
===========
Id     AdminStatus  OperStatus   Type
-----  -----------  -----------  -------------------
11863  [up]         [connected]  Statistics database
```

```
APIVersion ConfiguredURI    ConnectedURI     Failures LastTimeConnected
---------- -------------    -------------    -------- -----------------
[2]        tcp://10.10.10.1 tcp://10.10.10.1 1        20100325_153719
```

The `show service spb connections` CLI command does not show the IP of the SPB until a connection is established. Instead it shows 127.0.0.1.

To revert to the high-security configuration, remove the `tcp://` prefix from the SPB IP address.

4.4 Troubleshooting Subscriber Mapping

You can monitor the subscriber-to-IP-address mappings cache using the `show service subscriber-management dashboard` CLI command.

The dashboard runs continuously, displaying running totals for variables such as mapping and un-mapping rates, and the current number of mapped and un-mapped subscribers in the system. Some statistics of particular interest include:

- LookupsTimedOut counter, indicates that there was a delay from the SPB during a subscriber lookup which could indicate the SPB is becoming overloaded.
- IpNotInSubscriberClass counts subscribers found in a network class defined in subnets.txt but that have not been defined in subscriber_classes in policy.conf.

See the *SDE CLI Reference Guide* and *Subscriber Mapping User Guide* for additional information.

5

Subscriber Statistics Collection

5.1 Subscriber Statistics Overview

Subscriber statistics collection can be used to track the traffic statistics on a per-subscriber basis.

To use subscriber statistics, subscriber-IP mapping must first be in place.

Subscriber statistics can be telescoped at these levels:

- basic: traffic statistics are recorded on a per-subscriber basis
- protocol: traffic statistics are recorded on a per-subscriber, per-protocol basis

The subscriber statistics action may specify for which node statistics are to be collected: “client”, “server”, “sender”, or “receiver”. Note that in rule conditions, this information is referred to as the node qualifier.

5.2 Configuring Basic Statistics

Basic Rx and Tx statistics are recorded against all active subscribers for each logging interval.

Use the `count subscriber basic` action to enable basic subscriber statistic collection in SandScript. For example:

```
if true then count subscriber basic
```

Basic Rx and Tx statistics are recorded against all active subscribers for each logging interval. When enabled, all statistics are logged if either Rx or Tx is a non-zero value. For example, for a specific subscriber, if 100,000 bytes are sent but no bytes are received, a record is written to the database indicating 0 Rx bytes and 100,000 Tx bytes.

You can specify filters such that a minimum amount of traffic is sent by a subscriber before the database record is written. Any statistics below both minimums are filtered out. Setting filters reduces the amount of information logged to the database.

The CLI configuration commands are:

CLI command	Description	Valid Values	Default
set config service statistics subscriber minimum-bytes tx	Sets the minimum number of transmitted subscriber bytes needed to log to the database. Value of 0 indicates log all statistics.	0 - 2147483647	0
set config service statistics subscriber minimum-bytes rx	Sets the minimum number of received subscriber bytes needed to log to the database. Value of 0 indicates log all statistics. **ote:** The `set config service statistics subscriber minimum-bytes tx` command must be specified before this value will be considered in the logging decision.	0 - 2147483647	0

The Tx bytes filter is always required. This truth table illustrates how the filters affect statistics logging:

RxFilter	TxFilter	Rx	Tx	Result
0	0	0	0	LOG
0	0	1	1	LOG
0	0	0	1	LOG

RxFilter	TxFilter	Rx	Tx	Result
0	0	1	0	LOG
0	1	0	0	SUPPRESS
0	1	1	1	LOG
0	1	0	1	LOG
0	1	1	0	SUPPRESS
1	0	0	0	LOG
1	0	1	1	LOG
1	0	0	1	LOG
1	0	1	0	LOG
1	1	0	0	SUPPRESS
1	1	1	1	LOG
1	1	0	1	LOG
1	1	1	0	LOG
2	1	1	0	SUPPRESS

Where:

- RxFilter - the value set in `set config service statistics subscriber minimum-bytes rx`.
- TxFilter - the value set in `set config service statistics subscriber minimum-bytes tx`.
- Rx - bytes actually received by subscriber.
- Tx - bytes actually transmitted by subscriber. Tx filter is required, so if Tx filter is 0, PTS will always log.
- Result - one of:
  - LOG - data is logged to database.
  - SUPPRESS - data is not logged.

5.3 Use Cases

These examples are rules for generating subscriber statistics.

In the examples, a backslash (\) at the end of a line indicates that the rule, condition, or action is continued on the next line (carriage return escape).

The basic syntax to count statistics is:

```
if <condition> then count {node} subscriber basic|protocol
```

This rule counts basic statistics for all flows:

```
if true then count subscriber basic
```

Where:

- `true` - indicates that there is no condition and that this rule should always be executed.
- `count client subscriber basic` - performs a count on the client node to collect basic subscriber statistics.

If count subscriber basic is specified, by default the PTS will only publish the data once every hour. The default stat logging interval in seconds for subscriber basic statistics is configured using the `set config service statistics log-interval subscriber` CLI command.

This rule counts HTTP statistics for subscribers running HTTP servers:

```
if protocol "http" then count server subscriber protocol
```

Where:

- protocol “http”: is the condition.
- count server subscriber protocol: perform a count on the server node only to collect subscriber statistics against both subscriber and protocol.

6

Top Talkers

6.1 Top Talkers Overview

The Top Talkers feature identifies the top n subscribers based on upload or total (uploaded + downloaded) byte counts for all subscribers in either one or all clusters.

Subscriber flows are analyzed using rules in the `policy.conf` file for the cluster. Rules define the type of Top Talkers search to be run and the actions to be taken based on data collected by detailed user statistics.

One instance of Top Talkers is run per database. The module is deployed on an external statistics server such as the Sandvine Reporting Platform (SRP 3000). For detailed information on configuring Top Talkers, refer to the *SPB Administration Guide*.

Subscriber awareness whereby IP addresses are linked to specific subscribers is a prerequisite for identifying Top Talkers. For information on subscriber mapping, refer to the *Subscriber Mapping User Guide*.

6.1.1 Configuring Top Talkers

To run Top Talkers:

1. On the database server where Top Talkers has been configured, verify that the Top Talkers process is enabled and that a `policy.conf` file has been implemented to identify Top Talkers. For detailed information on configuring Top Talkers, refer to the *SPB Administration Guide*.

2. On the PTS element, create policies to collect basic statistics for all the users.

3. On the PTS element, create policies which indicate what to do with the Top Talkers information such as detailed statistics.

Example:
This excerpt from a `policy.conf` file illustrates policies to identify subscriber network classes, set attributes, and collect statistics.

```
# Outline the subscriber network classes
# These should be your "internal" network classes and subscriber classes
# names should match the network class names in subnets.txt file
# Detailed and basic user statistics will be collected for users present
# in these network classes only.
subscriber_classes "internal-dsl" "internal-cable" "internal-dialup" \
  "dsl-home" "dsl-business"

# Possible attributes that might be setup in the database for top
# talkers
# The top-talker process running on the database will be setting these
# attributes
attribute "top_talker" values "true" "false"

# For top talker subscribers - collect detailed (protocol level)
# statistics
# For all the other subscribers, collected basic statistics only.
# Note:  Based on basic (total upload and download bytes only),
# top-talker process on the database will set the "top_talker"="true"
# flag for the users.
# PTS then reads the database and if the "top_talker" flag is set for a
# particular user, then detailed statistics are collected for that user.

PolicyGroup
{
 if subscriber (expr(Flow.Subscriber.Attribute.Top_Talker="true" \
      then count subscriber protocol
 if true then count subscriber basic
}
```

```
# This line enables PTS physical network interface-based reporting
# (actual network interfaces present in PTS unit)
if true then count

# This line will configure PTS network class based reporting. It will
# record traffic flowing between various network classes present in
# subnets.txt file
if true then count demographic
```

6.1.2 Collecting Subscriber Statistics Using an Include File

An include file contains SandScript policy that can be accessed by referencing the include file in the local policy file or centralized policy file for the cluster.

An include file is available to collect basic subscriber statistics for everyone and detailed subscriber statistics for Top Talkers. For more information on using include files, see *Include Statements* on page 29

To collect subscriber statistics using an include file:

1. On each element, edit the local policy.conf file or edit the centralized policy file for the cluster.
2. Add this line to the file:

   ```
   include "/usr/local/sandvine/etc/policy.conf.top_talkers"
   ```

3. Reload PTSD by running the command: `PTS> reload`

7

Shaping

7.1 Configuring Shaping

Shaping manages internet traffic by constraining specific types of traffic to a limited amount of bandwidth.

Shaping values are applied to the cluster. You create a policy for the aggregate amount and apply this policy to all elements in the cluster. For example, if you want to achieve a shaping value of 100MB for the cluster, the shaping value on each element must be 100MB (for example, if there are two elements in the cluster, a shaping value of 100MB is applied to each element). For more information refer to *Rate Distribution* on page 110.

Up to four shapers can simultaneously shape a flow. Alarm model 23: Policy error, is triggered if SandScript exceeds this limit.

7.1.1 Shaper Definition

A shaper action must reference a shaper definition.

A shaper definition consists of some parameters followed by up to four priority definitions, each of which is followed by up to sixteen channel definitions. Each shaper is assigned a unique name. Shaping requires that a node qualifier be used along with the direction of the flow.

```
shaper "<ShaperName>" <rate> {strict true|false} {distribution none|simple|dynamic}
{max_carryover <%>} {<priorities>}
```

Where:

ShaperName Name of the shaper, such as "Http_c". The shaper name must:

- Start with a letter and contain only letters, digits, and underscores.
- Cannot be longer than 191 characters.
- Must be enclosed in quotation marks.

<rate> The rate for this shaper. Rate can be a number followed by units, the word "infinity" or a PAL expression. For example: 20Mbps, 50kbps, or `controller.CMTS_CONTROLLER.output.current`.

strict Indicates if the shaper rate is a strict maximum to be respected at all times. In general, the distribution of traffic to different modules may not be precise. As a result, it can be a little below or above the shaper rate. If the strict option is set to "true", the output rate will always be less than or equal to the shaper rate. The default value is "false".

distribution Indicates how the enforcement of the shaper rate is distributed to each module. The default is "dynamic" for all shapers except for unique-by shapers, which have a default of "none". The options are:

- simple: traffic enforcement is divided evenly amongst the module regardless of input traffic for each.
- none: traffic for each shaper instance is on only one module, so each module enforces the full rate. The shaper module assumes that traffic has been directed to one module via load balancing; it does not validate this assumption.
- dynamic: traffic rates are continually reassessed, and enforcement is divided among the modules according to demand.

max_carryover The maximum percentage by which the shaper rate can be exceeded in order to smooth the output rate of the shaper to the specified rate over time. Default is 10%. If the value is set to 0%, any unused bandwidth will not be used up in the future and will be lost.

7.1.2 Shaper Priorities

A maximum of four priorities can be defined for a shaper.

Traffic for a higher priority always takes precedence over traffic for a lower priority. The priority levels are determined by the order in policy (first priority is the highest).

The syntax is:

```
priority "<PriorityName>" {max_rate {<%>|<n>}}{min_rate {<%>|<N>}} {depth <depth>
{slots|bytes|ms}}{algorithm {shape|police|virtual}}{channels}
```

Where:

PriorityName The name of the priority

max_rate Maximum traffic rate permitted. Where % is a percentage of the total shaper rate, or N is a rate expression consisting of a number followed by one of these units: bps, kbps, Mbps, bytes/s, kbytes/s, Mbytes/s. For example: 100bps or 100bytes/s.

A max_rate on a priority defines a target for that priority's traffic along with the priorities above it. The shaper tries to meet that target by affecting only the traffic of the priority it is associated with. This means that the priority with the max_rate is only permitted bandwidth if the sum of priorities above it is less then the max_rate. At this point, the priority with the max_rate is permitted to pass through the difference of traffic, up to max_rate.

min_rate The minimum of traffic that is guaranteed by this shaper. Where % is a percentage of the total shaper rate, or N is a rate expression consisting of a number followed by one of these units: bps, kbps, Mbps, bytes/s, kbytes/s, Mbytes/s. For example: 100bps or 100bytes/s.

The min_rate feature is meant to be applied on the lowest priority traffic to ensure it isn't shaped out of existence. Therefore, a single min_rate can be used on a shaper with a max_rate, or multiple min_rates can be defined on a shaper without a max_rate defined. If you define multiple min_rates for a shaper with a max_rate then the aggregate shaper output is under- or over-shaped.

depth Maximum queue size in slots, bytes, or ms. Depth does not apply for policing. For more information, see *Queue Depth* on page 104. Example: depth 50 slots, depth 1000 ms.

Minimum size is 4500 bytes or 3 slots. If using ms, the 4500 byte minimum limit still applies. The maximum size permitted is 1 Mbyte or 2048 slots. If minimum or maximum values are exceeded, the value is automatically moved to within the permitted range. If a value is not specified, the default is either 50 slots or 5ms, whichever is larger.

algorithm Indicates how to shape the priority. Note that "shape" is the default. If policing or virtual queues are to be used, the algorithm setting can be used. For details on policing and virtual queues, see *Optimizing Performance* on page 109.

Note that rate, max_rate, min_rate, and depth (in bytes only) can be integers or have decimals. However, all values are rounded to the nearest byte/bps. If a value greater than four billion is required, a decimal must be used (for example, 5.0 Gbps works, but 5Gbps does not work).

Each priority level can be given a max_rate value, which is the maximum rate for that priority minus the sum of the rates of all higher priorities. For example, if the sum of the bandwidth of all priorities above a priority with a defined max_rate is greater than the max_rate value, then that priority with the defined max_rate value is permitted no bandwidth whatsoever. If max_rate is a percentage, it is a percentage of the shaper rate. This means that

```
shaper "Shaper1" 50Mbps priority "normal" max_rate 50%
```

is equivalent to

```
shaper "Shaper1" 50Mbps priority "normal" max_rate 25Mbps
```

Each priority level can be given a min_rate value, which guarantees a minimum amount of traffic from a low priority, preventing the higher priorities from completely dropping all lower-priority traffic.

max_rate is used to permit the combined rate of priorities to a certain rate and is not intended to permit a single priority to a limit. max_rate is generally applied to lower priorities and often in conjunction with an infinity rate shaper.

This example shows how to effectively use min_rate and max_rate parameters:

```
shaper "test" infinity \
    priority "high" \
    priority "low" max_rate 100Mbps min_rate 10Mbps
```

In the above example, the shaper is effectively capped at 100Mbps, unless the high priority (which has no limit) bridges more than 100Mbps. In this case, "low" is allowed only to bridge its min_rate.

Note:
min_rate of various priorities are calculated first and takes priority when distributing traffic. Sandvine recommends you not to use max_rate on high value priorities. Use max_rate only on low value priorities.

7.1.3 Shaper Channels

The syntax is:

```
channel "<ChannelName>" weight "weight"
```

Where:

channel a priority queue that has a specific weight. There is a limit of 16 channels per priority. The traffic for a shaper is divided amongst the channels proportionally. For example if channel_A has a weight of 1 and channel_B has a weight of 2, one third of the traffic will be handled by channel_A and two thirds of the traffic by channel_B.

weight weight that can be specified for a particular queue. Particularly for non-aggregate use cases, use the smallest practical value that will suffice (for example, do not use 90 and 10 if 9 and 1 suffice). Maximum queue weight is 1000.

7.2 Shaping Considerations

When using shapers, you should be aware that:

- Aggregate shaping is preferred over fair shaping as it provides better performance and less memory is used.
- Use a different shaper for each packet direction.
- Explicitly specify queue sizes in latency, bytes, or packets (latency is in milliseconds (ms)).
- Shaping bandwidth can be specified as any value between 78.125 bps and approximately 335.5 Gbps, however, it is applied in multiples of 78.125 bps. The actual value is rounded down to the nearest increment. A requested shaping bandwidth less than the granularity (i.e. between 0 and 78.125 bps) is rounded up to 78.125 bps.
- Setting the shaping bandwidth to zero is the same as setting it to the minimum supported shaper rate, which is 78.125 bps.
- For shapers which have many queues (as a result of per-subscriber shaping or subscriber fair shaping), the queue depth should be set to a small value (for example, 5 slots) to prevent running out of memory.
- Some protocols take more than one packet to be recognized by the PTS. To ensure that a high priority flow is not shaped incorrectly, the shaping action is not applied until after the flow has been recognized.
- If a large bundle is rebalanced (due to PPU overload, for example), and traffic is getting shaped minimally on the new PPU, the new traffic will be significantly shaped momentarily until the system converges to a steady state.

7.2.1 Effect of Queue Size on Latency

The queue length specified for a shaper dictates the **maximum** latency introduced by the shaper.

Note that for a "shared-by" shaper, this is the maximum latency for a particular queue. (For more information on shared-by, see *Creating a Shaping Action* on page 105.) The effective maximum latency for any packet is a function of the number of contending consumers, and traffic.

When using a virtual shaper, the maximum latency for a shared by shaper is for all the queues, not for each one as with regular shaping. When using policing, latency is zero as all packets are either dropped immediately or sent immediately.

If traffic is non-bursting, and below the target bitrate, there is zero latency introduced by the shaper, as packets are dispatched immediately, without any queuing. Excess rate traffic will be queued, with the latency of a packet dictated by the time of queuing, and the size of the packets already in the queue.

For example (slightly simplified), assume a 4Mbps non-shared shaper, receiving 500 byte frames. The expected interframe delay is 1ms (4Mbps/(500 bytes * 8 bits/byte)). If a packet arrives, and the shaper has been idle, the packet is immediately sent, with no additional latency. Call this time t0=0. If a subsequent packet arrives, say at time t1=0.5ms, it is early and needs to be queued. It is scheduled to be sent at time t2=1.0ms. So this packet experiences 0.5ms of introduced latency, to smooth out the flow of packets. If we assume this shaper has a maximum queue length of 50 slots, the maximum latency that can be introduced is 50ms. If the maximum queue length had been specified in bytes, say 5000 bytes, this would correspond to maximum latency of 10ms.

Bear in mind that any latency specification is constrained to the configured default minimum and maximum sizes:

- 1000ms of maximum latency at 1 Mbps corresponds to 125000 bytes, which is less than the default maximum.
- 1000ms of maximum latency at 100 Mbps corresponds to 12,500,000 bytes, which would be constrained to the default maximum of 1,048,576 bytes, or, effectively, ~84ms of maximum latency.

7.2.2 Queue Depth

Setting the queue depth to the right value is important for ensuring that bursty traffic is smoothed out, channel weights are respected, and fairness is achieved for fair shapers.

Setting the depth too low can compromise either or all of these goals. However, larger queue depths use more resources, as there is a limited number of packets that can be queued at once. Also, larger queues introduce more latency than smaller queues. These guidelines will help determine the correct queue depth for a particular application:

- For regular shaping, the default of 50 slots is usually ideal. If channel weights are not being achieved even though there is sufficient traffic to both channels, a larger queue depth may be needed (for example, 100 slots). When shaping aggressively (i.e. shaper rate is much smaller than the input rate), a smaller queue depth can be used (for example, may be necessary to reduce latency).
- For per-user shaping and fair shaping, queue depths should be set small (for example, 5 slots). Since there are usually many queues in this scenario, each one must be smaller to avoid exhausting the memory. If the total number of queued packets per PTS 24000, 22000, or 14000 exceeds 500 000, there is a greater risk of running out of packet memory. Run `show system resources` command in the CLI to show the current packet memory allocations. If packet memory is exhausted, some traffic will be lost and policy may not function correctly. If this happens, reduce the queue depth or use a different shaping algorithm. The minimum queue depth is 3 slots.
- When using virtual queues, the queue is shared for all fair-shaping instances and channel specifications. To achieve the defined channel weights and fairness, the queue depth must be fairly large. A rule of thumb is to allocate 400 slots for priorities with virtual queues that are shared by subscriber.

Note:
For policing, packets are not queued and therefore it is not necessary to configure queue depth.

7.3 Creating a Shaping Action

A shaping action references a shaper and the node-qualifier to shape.

The syntax is:

```
shape to|from <node> shaper "<ShaperName>"
{priority "<PriorityName>"} {channel "<ChannelName>"} {unique by <expr>}{shared by <expr>}
```

Where:

"ShaperName"	The shaper to use.
node	Can be one of client|server|sender|receiver|subscriber|internet|source|destination. It indicates the direction in which packets are shaped. For example, shape to client indicates that only packets travelling from server to client are shaped.
"PriorityName"	If the shaper defines priorities, the "PriorityName" must correspond to a name specified in a shaper definition, otherwise, the priority clause must not be present in the action. If the specified priority does not exist, an error is generated.
"ChannelName"	If the shaper defines channels, the "ChannelName" must correspond to a name specified in a shaper definition, otherwise, the channel clause must not be present in the action. If the specified channel does not exist, the policy will fail to load.
unique by and shared by	Determines whether the bandwidth allocated by the shaper is allocated totally to the user instance or is shared across all instances. • When SandScript expressions are used, the entire expression list should be enclosed in parentheses. • If non-SandScript expressions are used, parentheses are not required. • Do not mix SandScript expressions and non-SandScript expressions in the same list.

Note:
Although the brackets appearing in this section are occassionally not necessarily needed, Beat Practice indicates that there constant use will prevent errors that are difficult to identify or troubleshoot.

Note that:

- Multiple expressions can be used by separating them by a comma. For example, a shaper that is unique by internal-ip, protocol means that for every internal-ip, protocol pair, there is a unique shaper instance.
- If multiple actions reference the same shaper, they must be consistent with each other in terms of unique by. That is, if one shaper action is unique by something, all other shaper actions for the same shaper must also be unique by the same thing, otherwise an error will be generated. No error will be reported if both are unique by different things (for example, one is unique by ip, another is unique by sub), but this will produce undefined behavior.
- Likewise for shared by, except that it applies to a priority, not a shaper. This means that you cannot have two actions which apply to the same priority with inconsistent shared by clauses. You can however have inconsistent shared by clauses for two different priorities, each from the same shaper.

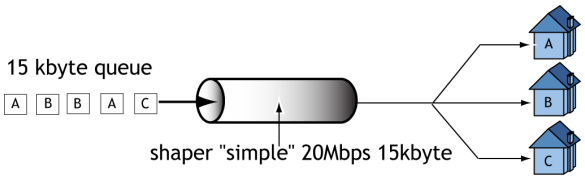
Example:
For example:

```
shaper "PerSubscriber" 50kbps depth 5Kbytes
if expr(Flow.ApplicationProtocol = Protocol.HTTP) \
then shape to subscriber shaper {"PerSubscriber" unique by Session.Id}
```

7.4 Use-cases

7.4.1 Simple Shaping

This example shapes HTTP download traffic (and upload to internal servers) to an aggregate total of 20Mbps. Note that the allocation of bandwidth to each flow or IP is not equitable.



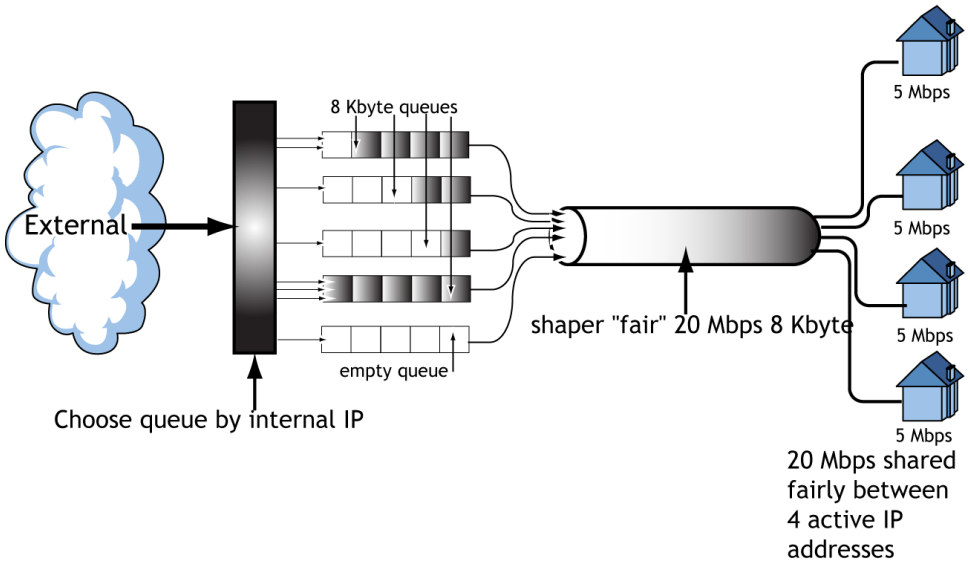

```
shaper "Simple" 20Mbps depth 15Kbytes
if expr(Flow.ApplicationProtocol = Protocol.HTTP) \
then shape to subscriber shaper "Simple"
```

7.4.2 Fair Shaping

This example shapes HTTP upload traffic (and download to internal servers) to an aggregate total of 20Mbps.

The allocation of bandwidth to each IP is fair as all active IPs share the available bandwidth equitably. This type of shaping can be resource intensive as there is an individual queue for each IP address.

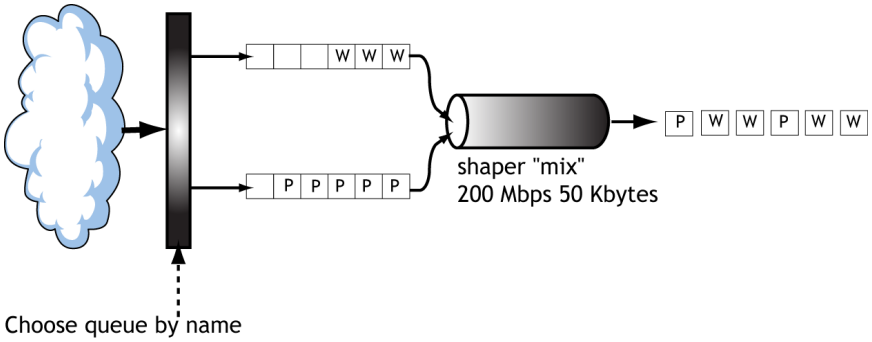
For fair shaping to work correctly across multiple modules in the PTS each value in the shared-by key must exist only on one module. This means that the shared by parameter must match the load balancer bundle definition or be a subset of the load balancer bundle. For example, if a shaper priority is shared by subscriber, subscribers must not be divided across modules; load balance by subscriber. If the shaper is shared by IP, load balance by IP.




```
shaper "Fair" 20Mbps depth 8Kbytes
 if protocol "http" then shape to internet shaper "Fair" shared by internal-ip
```

7.4.3 Aggregate User Shaping

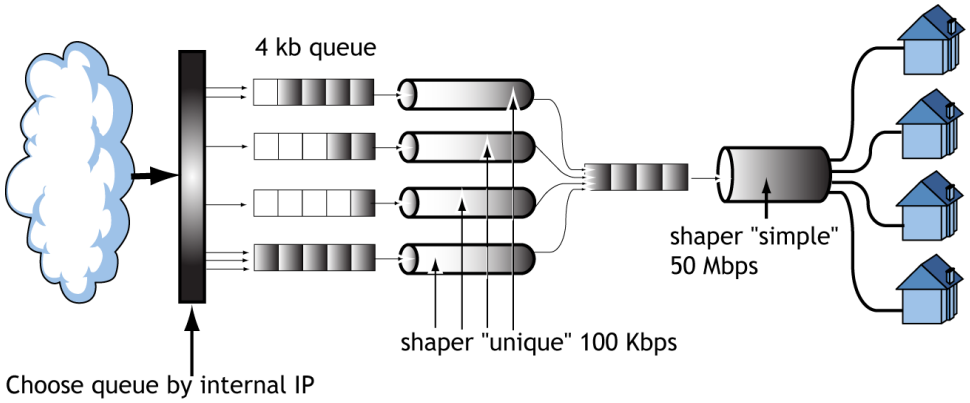
This example shapes HTTP download traffic (and upload to internal servers) to 50Kbps for each internal user.




```
shaper "PerSubscriber" 50kbps 4Kbytes
if expr(Flow.ApplicationProtocol = Protocol.HTTP) \
then shape to subscriber shaper "PerSubscriber" unique by Session.Id
```

7.4.4 Multiple Shapers

This example shapes HTTP download traffic (and upload to internal servers) to 100Kbps for each internal user but limits the total to 50 Mbps aggregate.



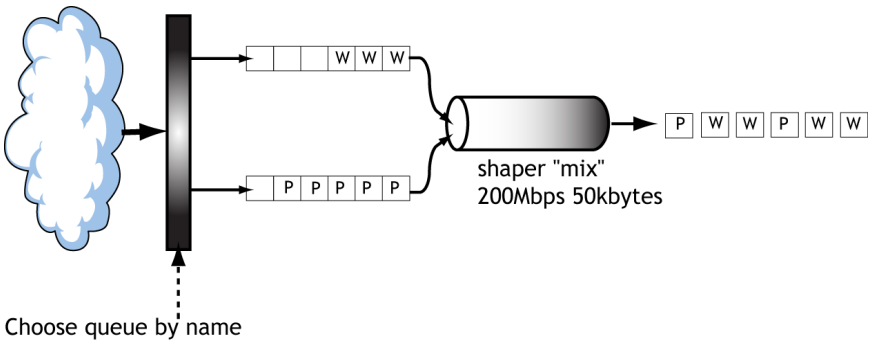

```
shaper "Different" 100Kbps depth 5 slots
shaper "Simple" 50Mbps depth 15ms

if expr(Flow.ApplicationProtocol = Protocol.HTTP) then shape to subscriber \
 shaper "Different" unique by Session.Id and shape to subscriber shaper "Simple"
```

Note the use of the ms notation in the simple shaper definition. This provides a convenient way of calculating the queue size based on the rate. In this example, the queue size will be 94kbytes guaranteeing a maximum latency of 15ms (queue size is calculated as (50 Mbps/8 bits/bytes)*0.015 seconds = 94 kbytes).

7.4.5 Traffic Prioritization

This example shows a single shaper with multiple priority levels.

Here P2P traffic is given a lower priority traffic than web, which means that any web traffic will be sent before any P2P traffic. In other words, whenever a packet is ready to be sent, a P2P packet is only sent if there are no queued web packets.



This introduces the potential to starve queues, that is, if there is enough web traffic to use the whole shaper rate, no P2P traffic will get through the shaper.

```
shaper "Mix" 1Gbps \
 priority "Web" algorithm shape \
 priority "P2P" algorithm shape

if protocol "http" then shape to subscriber shaper "Mix" priority "Web"
```

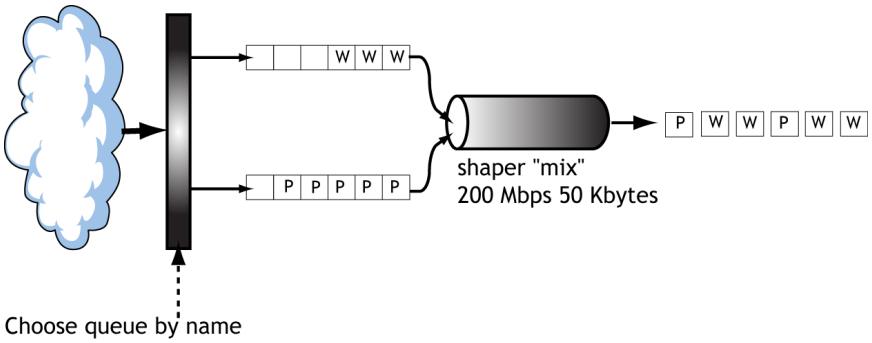
```
if (protocol "bittorrent" or protocol "gnutella") then \
 shape to subscriber shaper "Mix" priority "P2P"
```

7.4.6 Specific Channels and Weighted Fair Queuing (WFQ)

This is an example of a single shaper with multiple traffic channels (queues).

Weighted fair queuing (WFQ) can be achieved using traffic channels. This example shows HTTP and P2P traffic sharing a 200 Mbps shaper with more importance given to the HTTP traffic. If both HTTP and P2P traffic are present, HTTP traffic will be forwarded at twice the rate of P2P traffic.

The weights that are specified by the shaper indicate the relative importance. If one channel is assigned a weight of 2 and another channel is assigned a weight of 1, then the channel 2 weighted pipe takes twice as much traffic in bytes as the channel with the weight of 1.




```
shaper "Mix" 200Mbps depth 50Kbytes channel "Web" weight 2 channel "P2P" weight 1

if protocol "http" then shape to subscriber shaper "Mix" channel "Web"
if (protocol "bittorrent" or protocol "gnutella") then \
 shape to subscriber shaper "Mix" channel "P2P"
```

7.5 Optimizing Performance

It may be possible to handle more subscribers with the same equipment by policing traffic and using virtual queues.

Configuring rate distribution reduces the processor usage which may reduce the impact of shaping on the amount of traffic the PTS can handle.

7.5.1 Policing Traffic

Traffic policing improves the performance of shaping by deciding if a packet should be dropped or sent when it arrives and then dropping or sending it immediately.

The advantages over regular shaping are:

- There is no latency since packets are never queued.
- Memory usage is reduced (no need to store queued packets), so more unique instances can be supported (for example, if unique by IP, more IPs can be supported).

Policing has these disadvantages:

- Since packets are not queued, the bandwidth is not smoothed out (for any time interval, the output rate can be no larger than the input rate).
- Policing cannot be combined with named channels (specific queues) or with shared-by.

Each priority level is either a shaper or a policer. A single shaper can have multiple priorities, some of which are shapers, and other policers. An example of a policing policy is:

```
shaper "Web" 5kbps priority "Normal" algorithm police
if protocol "http" then shape to subscriber \
 shaper "Web" priority "Normal" unique by Session.Id
```

7.5.2 Virtual Queues

Virtual queues are an optimization which apply mostly to fair shapers.

Normally, with a fair shaper, each IP has its own channel, which is a queue of packets for that IP. By making these queues virtual, there is only one queue, instead of one for each IP. Each IP's channel then acts like a policer: as a packet arrives, a decision is made whether to enqueue this packet into the common queue or to drop it.

Virtual queues have these advantages:

- There is less latency since the total number of queued packets is smaller (one big queue instead of many medium sized queues)
- Memory usage is reduced (fewer queued packets), so more subscribers can be supported.

Virtual queues have a single disadvantage. The fairness of the fair shaping is less precise. That is, in some cases, the shaper approximates fairness, rather than being perfectly fair. However, in most cases, there should be no difference.

When using fair shaping, it is strongly recommended that virtual queues be used. For fair shapers with virtual queues, the queue depth should be quite large (for example 500 slots) to ensure maximum fairness.

7.5.3 Rate Distribution

A shaper's distribution parameter provides these options: none, simple, or dynamic.

The shaper rate is a per-cluster rate. Since the traffic is not necessarily balanced to all modules evenly, the PTS adjusts the shaper bandwidth for each module according to its share of the load. For example, if one module gets twice as much traffic as another, the output rate of the first module will be twice that of the second.

Dynamic

The default distribution option is dynamic, except for a unique-by shaper where the default setting is "none" (see below). Since traffic is constantly changing, the PTS continually evaluates the load on each module and re-adjusts the individual module rates accordingly. This consumes some system resources, and there are cases where it is not necessary.

None

The "none" distribution option can be used when all shaper instances exist on exactly one module. For example, if the load is balanced by IP and a shaper is unique by IP, each unique by instance appears on exactly one module. In this case, the keywords 'distribution none' can be used to indicate that the shaper rate should not be divided amongst multiple modules, but that the full shaper rate will be given to the single module with that IP.

Using the keyword 'none' when an instance's traffic appears on multiple modules means that the output rate will not be respected.

An example of the none distribution option is:

```
shaper "Good" 100kbps distribution none
if true then shape to internet shaper "Good" unique by internal-ip
```

The following is an example where the none option should not be used. Because this example is an aggregate shaper, each module will output 100kbps, rather than 100kbps across the cluster.

```
shaper "Bad" 100kbps distribution none
if true then shape to internet shaper "Bad"
```

Simple

Distribution can be optimized when load is evenly balanced to the different modules. In this case, the shaper rate should be evenly distributed to the modules. This can be achieved with the "simple" option.

If it is known that the load is evenly balanced, each module should output 1/N times the shaper rate, where N is the number of modules. The "simple" keyword forces the rates to be divided evenly. The load is not checked and no adjustments are made.

The default behavior (dynamic) will always produce the correct result: shaper rates will be distributed to accommodate the load, however, in some cases, the performance can be improved by using either "none" or "simple" distribution.

8

Quality of Experience

8.1 Quality of Experience Reports

Quality of Experience (QoE) attempts to characterize, through reports, the performance of the service provider's network for specific protocols in specific network classes and for users within network classes.

The types of reports produced are:

- Flow bandwidth charts - For each network class and protocol, the distribution of bandwidth achieved by that protocol for the network class.
- Flow efficiency charts - For each network class and protocol achieved TCP efficiency. This can highlight protocols which are not efficiently using the available bandwidth.
- User bandwidth chart - For each network class, the distribution of bandwidth achieved for transfers to and from that network. For example, the dial-up network class will have less bandwidth than DSL classes.

The data for these reports is collected using a random sampling method. Only a relatively small portion of the flows are actually measured to produce the results. A higher sampling rate produces more representative results of the true network activity, but at a performance cost. The percentage of sampling can be set in the rules (sample-rate parameter). Also, the frequency that a selected flow is sampled can be set (sample-quantum parameter). For example, if the sample rate is 1% and the sample quantum is 5 seconds, then each chosen flow will be examined every 5 seconds for the purpose of bandwidth and efficiency calculations.

To produce QoE reports, a rule must be created for each of the desired reports.

8.2 QoE Histograms

Sandvine provides rules, available using the monitor-config policy, that you can use to create histogram reports.

The available histograms are:

- **flow bandwidth histogram** - Charts the number of flows that achieve instantaneous bitrates corresponding to bitrate ranges (bins).
- **flow efficiency histogram** - Charts the number of flows that achieve instantaneous TCP efficiencies corresponding to efficiency ranges (bins). The data collected for this report samples random flows for each network class and calculates the instantaneous efficiency for each sampled flow every sample-quantum period. The instantaneous bitrate is then tabulated in the bin that encompasses it. Once each period, the tabulated values for each bin are sent to the database for inclusion in the reports.
- **user bandwidth histogram** - Charts the number of users that achieve instantaneous bit ranges corresponding to bit ranges (bins). The data collected for this report samples random users (IP addresses) for each network class and calculates the instantaneous bitrate for each sampled user every sample-quantum period. The instantaneous bitrate is then tabulated in the bin that encompasses it. Once each period, the tabulated values for each bin are sent to the database for inclusion in the reports.

The syntax is:

```
monitor-config rule-number {configuration
 {sample-rate <n>}
 {sample-quantum <n>}
 {period <n>}
 {low-value <n>}
 {high-value <n>}
 {bins <n>}}
```

Where:

rule-number Specifies a network-protection rule provided by Sandvine in the `policy.sandvine.conf` file. Works with the configuration. Choose one of:

- 500 - for flow-bandwidth-histogram.
- 501 - for flow-efficiency-histogram.
- 502 - for user-bandwidth-histogram.

configuration Specifies the type of histogram. Works with the rule number. Choose one of:

- flow-bandwidth-histogram - for rule-number 500.
- flow-efficiency-histogram - for rule-number 501.
- user-bandwidth-histogram - for rule-number 502.

sample-rate Specifies the number of flows that will be sampled to generate the histogram. Setting this value much higher than the default of 1% will have a negative performance impact on the element.

Allowed values are 0 - 1.0. A value of 0.01 means that 1% of the flows are measured. Default is 0.01

sample-quantum Specifies how often each sampled flow has an instantaneous bitrate calculated. Allowed values are:

- s|seconds
- m|minutes
- h|hours
- d|days

For example: 15s, 15seconds, or 1d. Suggested value is 15 seconds.

period Specifies how often the collected information is written to the database for inclusion in the histogram reports. Allowed values are:

- s|seconds
- m|minutes
- h|hours
- d|days

For example: 15s, 15seconds, or 1d. Recommended value is 1 hour.

low-value For flow efficiency and user bandwidth histogram, the value in bps which represents the lowest bitrate represented in the histogram. Value will normally be 0.

For flow bandwidth, the lowest value in percent that is represented on the histogram. Value will normally be 0%.

high-value For flow efficiency and user bandwidth histogram, the value in bps which represents the highest bitrate represented in the histogram. This value should be approximately the expected maximum bitrate allocated to a subscriber (56000 bps for dialup, 3000000 for 3Mbit and so on).

For flow bandwidth, the highest value in percent that is represented on the histogram. Value is 100%.

bins For flow efficiency and user bandwidth histogram, the number of bins that collected bitrate values are sorted in. Each bin is `(high-value – low-value)/bins` bps wide. The default, and maximum recommended value, is 20.

For flow bandwidth, the number of bins that collected values are sorted in to. Each bin is `(high-value – low-value)/bins` % wide.

Example:

A flow bandwidth histogram example:

```
monitor-config 500 flow-bandwidth-histogram  period 3600 \
 sample-quantum 10 sample-rate 0.01 low-value 0 \
 high-value 3000000 bins 20
```

Example:
A flow efficiency histogram example:

```
monitor-config 501 flow-efficiency-histogram period 3600 \
  sample-quantum 10 sample-rate 0.01 low-value 0 \
  high-value 100 bins 20
```

Example:
A user bandwidth histogram example:

```
monitor-config 502 user-bandwidth-histogram  period 3600 \
 sample-quantum 10 sample-rate 0.01 low-value 0 \
    high-value 3000000 bins 20
```

9

Controller

9.1 Controller Overview

Policy controllers are used to create a closed loop control system.

The input parameter is quality of experience (QoE), which the controller measures and reports on. Based on the data collected, the controller can adjust traffic using a shaper; traffic being the output parameter of the closed loop system.

The quality metrics of the input parameter are defined through policy, along with metric options which configure how to provide metric values into the control system. Each metric is converted into a quality score from 0 to 100 (based on values and configuration parameters). Multiple metrics may be used in combination.

A maximum of one controller may be defined in the policy.

Note:
Controllers should be used with caution because of the behavioral complexity. They are provided pre-configured as part of Sandvine's Fairshare Traffic Management functionality.

9.2 Controllers

The syntax is:

```
controller "name" \
 minimum {minimum_output_value} \
 maximum {maximum_output_value} \
 interval {controller_sample_interval} \
 output_histogram {histogram_field} \
 ratio {controller_ratio_value} \
 metric "MetricName" metric_options \
  {metric "MetricName2" metric_options "MetricName3" metric_options ...}
 {unique by unique_by_spec}
```

Where:

“name” The name of the controller, used to identify the controller in CLI commands and when referencing the controller using SandScript. The name must be unique amongst controllers.

minimum The minimum allowed output of the control system. The controller output is not permitted to go below this value, even if the QoE is not maximized. Value is represented as an integer.

maximum The maximum allowed output of the control system. Value is represented as an integer.

interval Optional. A time interval which the control system will run on. It is the frequency at which the controller is evaluated.

This value should be chosen according to performance requirements and test results. Choosing a value that is very low will result in a higher performance cost. A higher value should only be considered where the lead/stabilization time are less important due to real-time requirements.

The minimum value is 15, and there is no maximum. Default is 30 seconds.

output_histogram Optional. Specifies a histogram which can be used to define the range of data to store from the control system. The expression MUST be a histogram SandScript field. See *Histograms* on page 71.

ratio The ratio at which to adjust output, relative to the current output. This parameter indicates what fraction of the current output to use as a step function.

For example, if the current output of the control system is 20M, and the ratio is set to 5, then the step amount is 5% of 20M, or 1M. Then, if the control system generates an output of 5, it will change the output of 5 *1M = 5M, which would then have a new output of 25M. A larger ratio will make the control system output more

coarse, resulting in a larger steady state error with the benefit of decreased lead time and decreased stabilization time.

Value is expressed as a real number. Default is 0.5.

metric At least one metric must be defined. Up to five can be defined for one controller. Each metric is individually evaluated and a score is calculated from the data that has been received. When multiple metrics have been defined, the score from each metric is weighted against the others to produce a final score which is used by the control system.

unique by Defines the key for which to create unique instances of a control system. The unique by spec must consist of only classifier expressions whose total size fits within 64 bits.

9.2.1 Metric Options

The syntax is:

```
metric "MetricName" \
  data {histogram_field} \
  benchmark {benchmark_quality_value} \
  percentile {percentile_measurement_of_quality} \
  weight {weight_of_the_metric} \
  min_samples {minimum_number_of_samples} \
  tolerance_factor {tolerance_value} \
```

Where:

“MetricName” The name of the controller metric, used to identify the metric in the CLI or when referring to the metric using SandScript. The name must be unique among metrics for a given controller.

data A histogram expression which defines separate logical bins for quality samples. The histogram definition should be representative of the expected range of values for the input metric.

benchmark A benchmark value for which to calculate a score from the samples that are read from policy. The benchmark represents the ideal sample given ideal operating conditions measured for the network. Therefore, if the system is running at its optimal best, its score would be 100.

percentile Optional. When analyzing the histogram, the control system forms a single value based on the data. This parameter chooses which percentile of the value to use.

weight Optional. The weight of this metric. Used to calculate the quality score when there are multiple metrics in a controller.

min_samples Optional. The minimum number of samples required for the metric to be considered valid during a sample interval. If the metric is invalid during a sample interval, then no score is calculated for the metric and it will not count against other metrics. If no metrics have a valid number of samples, it is equivalent to a quality score of 100.

tolerance_factor Used in combination with the benchmark value to derive a score from the collected metric values. The tolerance provides a means to be more or less aggressive on the measurement, relative to the score. For example, with a metric that is to be minimized with a tolerance_factor of 3 and a benchmark of 100 produces a score of 0 at a metric value of 300. For a tolerance_factor of 5 and a benchmark of 100, the score is 0 when the measurement is 500.

9.2.2 Controller Input

Provides an input value to a controller metric.

The syntax is:

```
controller.<ControllerName>{[KEY]}.input<MetricName>(value:expression)
```

Where:

- <ControllerName> is a controller defined in policy.
- <MetricName> is a metric of that controller.
- <expression> must be of the type integer or float.

9.3 Controller Example

This example creates a controller system to optimize RTT for a subnet based on a benchmark of 60ms.

Example:

```
#Location of the subscriber. This example policy does not set this value, it is expected
that it would be set by some other policy.
classifier "SubscriberCmts" type integer16
histogram "RttDistribution" custom (50,80,120,200,300,500,800,1200,4000,8000)
histogram "HttpDistribution" custom (0,20,60,120,200,500,800,1200,3000,6000)
controller "CmtsController" \
           minimum 1000000000 /*1Gbps*/ \
           maximum 20000000000 /*20Gbps*/ \
           interval 15 seconds \
           output_histogram Histogram.RttDistribution \
           ratio 2.0 \
           metric "Rtt" data Histogram.RttDistribution \
    benchmark 60 percentile 80 weight 2 min_samples 3 tolerance_factor 2 \
               unique by (classifier.SubscriberCmts)

#defininitions
publish "COutput" Controller.CmtsController.Output
publish "CDemand" Controller.CmtsController.Demand
publish "CScore" Controller.CmtsController.Score
shaper "CmtsShaper" rate Controller.CmtsController.Output.Current \
     distribution dynamic

#read in QoE for RTT
if expr(Flow.ApplicationProtocol.IsNew AND Flow.Subscriber.HandshakeRtt is not null \
 AND Flow.Classifier.SubscriberCmts is not null) then \
 controller.CmtsController[Flow.Classifier.SubscriberCmts].input.\
  Rtt(value:ToMilliseconds(Flow.Subscriber.HandshakeRtt))

# Downstream shaping on all flows where the location is known
if expr(flow.classifier.SubscriberCmts is not null) then shape to subscriber shaper
"CmtsShaper" unique by (Flow.Classifier.SubscriberCmts)
```

10

Maps

10.1 Map Overview

Maps define a set of items and provides the ability to test for an item's existence within the set.

Each item in the map can be associated with an optional label, which can be accessed when looking up a specific item in policy. Items defined within the map can contain a glob match character (*) which gives extra flexibility when defining items in a map.

The different map types are:

- String map allows for a map to contain any string.
- Host map provides options to perform normalized match of hostnames.
- URL map provides option to perform normalized match of URLs. For use cases where maps contain only full URLs.
- IP Address map is optimized to match IP addresses. For use cases where maps contain only IP addresses.

You can view the currently defined maps using the `show policy map` CLI command which gives aggregate, real-time statistics that exposes information such as number of total queries against the map, number of total matches, and more.

When using maps, note these limitations:

- Both the SDE and PTS support a maximum of 100,000 map entries across all system maps. SandScript policy will not load if you exceed this limit.
- The PTS supports a maximum of 10,000 entries with the glob-match character across all maps.
- The PTS 24000 and 22000 provides 80Mb of memory for map use, while the PTS 14000/8210 and SDE provide 40Mb of memory. Use the `show policy map` CLI command to monitor resource usage.

10.2 String Maps

The syntax is:

```
map "<name>" string {label_type string|integer} {case_sensitive true|false}
{escaping true|false} files <file_list>|values <value_list>
```

Where:

name A unique name for the map. Note that name:

- Must begin with a letter and can only contain letters, digits, and underscores.
- Must not be a policy reserved keyword.
- Is not case sensitive.
- Cannot conflict with other function names (user-defined and internal).

label_type Specifies optional labels along with the entries. Defaults to string.

case_sensitive Indicates if case sensitive match should be used. Defaults to false.

escaping Enables escaping certain characters in the patterns of policy maps. Defaults to false if not specified. Valid escape sequences are:

Escape sequence	Meaning
\#	hash
\	space
\t	tab

Escape sequence	Meaning
\r	carriage return
\n	newline
\\	backslash

files <file_list> A list of quoted file names separated by spaces. These files will be read to populate the map. They can be absolute path, or relative to `/usr/local/sandvine/etc/`. This is the recommended source for a string map whenever possible.

values <value_list> A list of quoted strings separated by white space that define the contents of the map.

10.2.1 SandScript Functions for a String Map

Use functions to operate on the map.

Each map function has this syntax:

```
Map.<Name>.<Function>
```

Where:

- Name: is the unique name given to the map in the map definition.
- Function: each SandScript function transforms input and finds a match according to optional parameters specified in the map declaration. The functions that are provided by a string map are:

Function name	Description
Contains(*string*)	Returns true if *string* is matched by a key/pattern in the map, compared as specified by *case_sensitive* parameter in the map declaration.
FindLabel(*string*)	Returns the label associated with a key/pattern, if *string* is matched by a key/pattern in the map, compared as specified by the *case_sensitive* parameter in the map declaration. The return type is an integer/string as specified by *label_type* parameter in the map declaration.
FindPattern(*string*)	Returns the key/pattern associated with a key/pattern, if *string* is matched by a key/pattern in the map, compared as specified by the *case_sensitive* parameter in the map declaration.

Example:
This example uses a map file called subscriberlist.txt:

```
00:30:48:30:0d:8a
00:30:48:30:0d:8b
00:30:48:30:0d:8c
```

To access the map file:

```
map "MyMap" string files "subscriberlist.txt"
```

We can use "MyMap" with the pre-defined measurement “Matches”:

```
measurement "Matches"
if expr(Map.MyMap.Contains(Flow.Client.Subscriber.Name) then \
increment measurement.Matches
```

10.3 Hostname Maps

The hostname map allows for some extra functionality and optimization when the keys/patterns in a map are all host names. Hostname maps must only contain lists of hostnames, if a URL is specified the configuration will fail to load.

An example failure is:

```
Invalid hostname in hostname map `Invalid host at entry 12 in file
'/usr/local/sandvine/etc/map.allowed_hostnames.txt' in map 'AllowedHostnames'` ``
```

The syntax is:

```
map "<name>" hostname {label_type string|integer} {case_sensitive true|false}
{normalize true|false} files <file_list>|values <value_list>
```

The same parameters as string maps apply, plus `normalize` performs normalized match if set to true.

Normalization allows identifying two syntactically different URLs that represent the same resource. Policy maps support normalizing URLs before doing a match so that two such different URLs can be matched. Normalization support includes removing percent-encoding and dot-segments for a URL before matching URL lists. For example, equivalent URLS are:

Unnormalized URL	Normalized URL
http://www.example.com/../a/b/../c/./d.html	http://www.example.com/a/c/d.html
http://en.wikipedia.org/wiki/URL-%2FPercentencoding	http://en.wikipedia.org/wiki/URL/Percent-encoding

10.3.1 SandScript Functions for Hostname Maps

The SandScript functions that are provided for a hostname map are:

Function name	Description
Contains(*hostname*)	Returns true if *hostname* is matched by a key/pattern in the map, compared as specified by *case_sensitive* parameter in the map declaration.
FindLabel(*hostname*)	Returns the label associated with a key/pattern, if *hostname* is matched by a key/pattern in the map, compared as specified by the *case_sensitive* parameter in the map declaration. The return type is an integer/string as specified by *label_type* parameter in the map declaration.
FindPattern(*hostname*)	Returns the key/pattern associated with a key/pattern, if *hostname* is matched by a key/pattern in the map, compared as specified by the *case_sensitive* parameter in the map declaration.
FindHost(*hostname*)	Returns the host, if *hostname* is matched by a key/pattern in the map, as specified by *normalize* and *case_sensitive* parameters in the map declaration, or null if the *hostname* isn't found.

Example:
This example uses a map file called hostnamelist.txt:

```
www.example.com
www.sandvine.com
```

The policy to access the map file is:

```
map "MyMap" hostname normalize true case_sensitive false files "hostnamelist.txt"
```

We can use "MyMap" with the pre-defined measurement “Matches”:

```
measurement "Matches"
if expr(Map.MyMap.Contains(Flow.Client.Stream.Http.Host) then \
increment measurement.Matches
```

10.4 URL Maps

The syntax is:

```
map "<name>" url {{case_sensitive true|false} {normalize true|false}
{resource_param_match_order exact|any}} {{files <file_list>}|{values <value_list>}}
```

The same parameters as for string maps apply. In addition, `resource_param_match_order` defines how parameters contained in URLs should be checked when attempting to determine if an input URL is contained in a map. If exact (the default), the map attempts to match all text following the “?” character in a URL as if it were a string. If any, it attempts to match URL parameters in any order in which they may appear.

For the input “www.sandvine.com/index.html?a=1&b=2”, the matches are:

Map entry	Resource_param_match_order	Match?
www.sandvine.com/index.html?a=1&b=2	exact	Yes
www.sandvine.com/index.html?a=1&b=2	any	Yes
www.sandvine.com/index.html?b=2&a=1	exact	No
www.sandvine.com/index.html?b=2&a=1	any	Yes

10.4.1 SandScript Functions for URL Maps

A URL map type has a set of policy functions used to operate on the map in policy.

The syntax is:

```
Map.Name.Function(hostname, resource)
```

Where:

- Name—Is the unique name given to the map in the map definition.
- Function—Includes any of these functions:

Function name	Description
Contains(*hostname, resource*)	Returns true if *hostname* matches a key/pattern in the map, as specified by the *case_sensitive* parameter in the map declaration.
FindLabel(*hostname, resource*)	Returns the label associated with a key/pattern, if *hostname* matches a key/pattern in the map, as specified in the *case_sensitive* parameter in the map declaration. The return type is an integer or string, as specified by *label_type* parameter in the map declaration. Returns null if no label is found.

Function name	Description
FindPattern(*hostname, resource*)	Returns the key/pattern associated with a key/pattern, if *hostname* matches a key/pattern in the map, as specified in the *case_sensitive* parameter in the map declaration. Returns null if there is no match.
FindHost(*hostname*, *resource*)	Returns the host, if *hostname* matches a key/pattern in the map, as specified by the *normalize*, *case_sensitive*, and *resource_param_match_order* parameters in the map declaration. Returns null if there is no match.
FindResource(*hostname*, *resource*)	Returns the resource, if *resource* matches a key/pattern in the map, as specified by *normalize*, *case_sensitive*, and *resource_param_match_order* parameters in the map declaration. Returns null if there is no match.
FindUrl(*hostname*, *resource*)	Returns the URL, if *URL* matches a key/pattern in the map, as specified by *normalize*, *case_sensitive*, and *resource_param_match_order* parameters in the map declaration. Returns null if there is no match.

Example:
For example:

We have urllist1.txt

```
www.example.com/
www.test.com/req=sample1
```

And urllist2.txt

```
www.testsample.com/test/*
www.testsample.com/request*
http://www.sandvine.com/example.html?a=1&b=2
http://www.example2.com/*
```

The policy to access these map files is:

```
map "MyMap" url normalize true case_sensitive false  resource_param_match_order exact files
 "urllist1.txt" "urllist2.txt"
```

We can use "MyMap" with the pre-defined measurement “Matches”:

```
measurement "Matches"
if expr(Map.MyMap.Contains(\
 Flow.Client.Stream.Http.Host, \
 Flow.Client.Stream.Http.Resource)) then \
 increment measurement.Matches
```

10.5 IP Address Maps

This map is optimized to match IP addresses, for use cases where maps contain only IP addresses (IPv4 and IPv6).

The syntax is:

```
map "<name>" ipaddress {label_type string|integer}
        {escaping true|false} {{files <file_list>}|{values <value_list>}}
```

Use functions to operate on the map in SandScript. Each function has the syntax:

```
Map.<Name>.<Function>
```

Where:

- Name - Is the unique name given to the map in the map definition.
- Function - Transforms input and finds a match according to optional parameters specified in the map declaration.

The functions provided for IP address maps are:

Function name	Description
Contains(*ipaddress*)	Returns true if *ipaddress* is matched by an IP address in the map.
FindLabel(*ipaddress*)	Returns the label associated with an IP address, if *ipaddress* is matched by an IP address in the map. The return type is an integer/string as specified by *label_type* parameter in the map declaration.

Example:
For example, to increment the IP address map:

```
map "MyMap" ipaddress files "subscriberIPs.txt"
if expr(Map.MyMap.Contains(Flow.Internet.IPAddress)) then \
increment measurement.Matches
```

To use the `FindLabel()` function with IP Address maps:

```
map "MyMap" ipaddress label_type string files "subscriberIPs.txt"
string "MyString" = Map.MyMap.FindLabel(IPAddress("140.211.166.64"))
if(MyString is not null)
{
  # use MyString here
}
```

10.6 Loading Map Contents

Whenever the properties of a map are changed (map type, data source, and so on) a full policy reload must be performed.

Additionally, if the source type for the map is “values”, a full policy reload must also be performed if the values in the map are to be changed. However, if the source type of the map is “files”, the content of the maps can be modified without a full reload, meaning that while the new map data is being loaded, policy continues to be applied until the new content is loaded, at which point the new map content is swapped seamlessly.

To perform a reload of maps contents when the source type is “files” at the command line, run:

```
PTS> reload -maps
```

```
SDE> svreload -maps
```

If this reload fails for any reason, policy continues to apply with the content of the map that existed before the reload was issued. However, if the map file is not corrected upon PTS system restart, the map will fail to load.

10.7 Map File Format

When the source of map contents is defined as files, each file is read in one line at a time.

Each line should be of the format: `<key> <optional_label>`. Where:

- key is the item that will be stored in the map. It may contain one or more occurrences of the glob-match character (*). Note that the validity of a key depends on the type of map.
  - For URL maps, <key> should be a valid URL and can optionally start with `http://`.
  - For hostname maps, <key> should be a valid host name, it can optionally start with `http://`.
  - For string maps, <key> can be any string.
- optional_label is an integer or string value that is associated with the value in the map. While adding a label is optional, if you specify an integer label, then you must include the value. A string label will default to an empty string. All optional_labels in the file must be the same type as the label_type parameter in the map declaration.

Example:
For example:

```
www.example.com 1 www.sandvine.com 2
```

10.7.1 Glob Matching

The key defined within maps may contain the glob match character (*) to give added flexibility when defining keys within a map.

One or more glob match characters may be used in a key. For example, if this key is specified within a map:

```
www.example*.com
```

Then any of these inputs will be matched by the map:

```
www.example1.com
www.exampleAAA.com
www.example.com
```

If you wish a particular pattern to be matched before all others, put it into its own map to be searched first. For example, say we have these patterns:

```
www.exam*.com
www.example*.com
```

Because of the glob match character, both these patterns will match the string “http://www.exampleAAA.com”. To match the more specific www.example*.com first, put it into its own map file, called MapSpecific, and the more general keys into MapGeneral. Then create policy to search the more specific map first:

```
Classifier “URL_count” type string
if true then set Flow.Classifier.URL_count = \
   Map.MapSpecific.FindLabel(Flow.Client.Stream.Http.URL)
if expr(Flow.Classifier.URL_count is null) then set Flow.Classifier.URL_count = \
   Map.MapGeneral.FindLabel(Flow.Client.Stream.Http.URL)
```

10.7.2 Normalization for Hostname and URL Maps

Normalization allows identifying two syntactically different URLs that represent the same resource. Policy maps support normalizing URLs before doing a match so that two such different URLs can be matched. Normalization support includes removing percent-encoding and dot-segments for a URL before matching URL lists. For example, equivalent URLS are:

Unnormalized URL	Normalized URL
http://www.example.com/../a/b/../c/./d.html	http://www.example.com/a/c/d.html
http://en.wikipedia.org/wiki/URL-%2FPercentencoding	http://en.wikipedia.org/wiki/URL/Percent-encoding

11

Configuring Limiters and Session Management

11.1 Limiters Overview

Limiters can be used to apply actions after a threshold (such as number of connections) is exceeded, or to define a control system to manage the amount of bandwidth on the network.

Limiter declarations specify:

- What is being limited/controlled.
- The value of a measurement.
- Active connections.
- Bandwidth.
- The value of a SandScript expression.
- The limit or target value, for example 100 active connections or 100 Mbps.
- The “scope” of the limit or target value.
- Aggregate or unique-by.
- Optional prioritization of traffic.

The CLI shows statistics for each limiter defined in policy (limit/target, current value, number of flows actioned by the limiter and so on). For details on using `show policy limiters` CLI commands, refer to the *Sandvine PTS Operations Reference Guide* for this release.

Typically, limiters are combined with the tcp_reset action to implement Session Management policies, but they are flexible enough to be used to control any action.

Session Management policy differs between the PTS 8210 and the PTS 24000, 22000 and 14000:

- PTS 24000/22000/14000: Limiting values are applied to the cluster. Create a policy for the aggregate amount and apply this policy to all elements in the cluster. For example, if you want to achieve a limit of 100 connections for the cluster, the limit on each element must be 100 connections (that is, if there are two 14000 elements in the cluster, a limit of 100 connections is applied to each element).
- PTS 8210: Limiting values are per element. This means that if you want to achieve a limit of 100 connections for the cluster, the sum of the limits for all elements in the cluster must be 100 connections. For example, if there are four elements in the cluster, a limit of 25 connections could be applied to each element or values such as 80, 10, 5, and 5 connections could be applied: for a cluster total of 100 connections.

Note:
Session management of traffic in layer 2 (MPLS) and layer 3 (GRE) tunnels is not supported.

11.2 Declaring Limiters

The limiter syntax is very similar to the measurements syntax.

Every limiter has an underlying measurement – this relationship is either stated explicitly (by referencing a separately defined measurement) or implicitly by specifying conditions that indicate what to measure.

To limit/control the value of a measurement:

```
limiter "<name>" limit {connections} measurement "<measurement_name>" \
{strict true|false} {distribution none|simple|dynamic} {max_carryover <%>}

limiter "<name>" limit {Mbps|kbps|bps|Mbytes/s|kbytes/s|bytes/s}
measurement "<measurement_name>" {strict true|false} \
```

```
{distribution none|simple|dynamic} {max_carryover <%>} \
{priority "<priority1>" priority "<priority2>" ...}
```

To limit/control bandwidth or limit the number of active connections, either aggregate or unique-by ip/subscriber (or something else):

```
limiter "<name>" limit connections {where condition(s)} \
{strict true|false} {distribution none|simple|dynamic} {max_carryover <%>} \
{unique by client-ip|server-ip|internal-ip|receiver-ip|sender-ip|subscriber}
```

To limit/control the value of a SandScript expression:

```
limiter "<name>" limit connections expr(SandScript expression) \
{strict true|false} {distribution none|simple|dynamic} {max_carryover <%>}

limiter "<name>" limit {Mbps|kbps|bps|Mbytes/s|kbytes/s|bytes/s} \
{strict true|false} {distribution none|simple|dynamic}{max_carryover <%>} \
expr(SandScript expression) {priority "priority1" priority "priority2" ...}
```

name The name of the limiter. Must be unique amongst measurements and limiters (cannot have a measurement called “Protocols” and a limiter called “Protocols”). Note that the name:

- Must begin with a letter and can only contain letters, digits, and underscores.
- Must not be a policy reserved keyword.
- Is not case sensitive.

limit The number of active connections allowed by a connection limiter, or the target rate/value of a bitrate/expression limiter. The units applied to the limit define the limiter type:

- Connections means it is a connection limiter.
- A bitrate unit (Mbps|kbps|bps|Mbytes/s|kbytes/s|bytes/s) means it is a bitrate limiter. Note that unique by is not allowed on a bitrate limiter.
- No units means it is limiting an expression with complex units (not connections or bitrate). When limiting an expression, the units must match the units of the expression.

measurement_name The name of the measurement to apply the limit to. Connection limiters must reference connection measurements, rate limiters must reference bitrate measurements, expression limiters must reference expression measurements.

condition Indicates what to measure. If conditions are omitted, true is assumed (the measurement and limit are applied to all flows).

strict Indicates if the limit is a strict maximum to be respected at all times. In general, the distribution of traffic to different modules may not be precise. As a result, it can be a little below or above the limit. If the strict option is set to true, the output rate will always be less than or equal to the limit.

distribution Indicates how traffic is distributed to each module. The default is dynamic for all limiters except unique by connection limiters, which cannot be distributed. The options are:

- simple - Traffic is divided evenly amongst the modules regardless of the input traffic of each.
- none - Traffic for each limiter instance is confined to one policy process instance per module, so each module's policy process instance gets the full rate. The policy process instances can be listed using the `show service load-balancer modules` CLI command.

  For example, an 81Mbps limiter on a PTS 14210 with 1 policy process instance per module and 4 modules shows a 81Mbps * 1 * 4 = 324Mbps cluster limiting rate. The same 81Mbps limiter on a PTS 22400 with 4 policy process instances per module and 2 modules shows a 81Mbps * 4 * 2 = 648Mbps cluster limit.
- dynamic - Traffic is divided amongst the modules according to demand.

max_carryover	The maximum percentage by which the limit can be exceeded in order to smooth the output rate of the limiter to the specified rate over time. If 0, any unused limit will not be used up in the future and will be lost.
unique by	Defines a unique instance. A value is measured and a limit is applied to each unique instance. Options are: client-ip|server-ip|internal-ip|receiver-ip|sender-ip| subscriber|(SandScript expression list). For a list of SandScript fields, refer to the *PTS SandScript Guide*. Note that you cannot use unique by on a bitrate limiter.
priority "Priority_name_1" priority "Priority_name_2" ...	List of named priority levels for rate and expression limiters. The order in which the priority levels are specified is important - traffic assigned to the first priority is limited first. If the target rate is still not achieved, then traffic assigned to the second priority is limited, and so on, until the target rate is achieved. Priorities only apply to rate and certain types of expression limiters. The most common use for priorities is to manage new flows separately from existing flows.
expr(SandScript expression)	A limiter can be used to apply a limit to or to control the value of a SandScript expression. For example, a limit can be applied to the sum of two measurements. The units of the limiter must match the units of the expression.

Note that:

- The special source/destination node qualifiers can be used to specify direction for bitrate measurements.
- When limiting an expression, the units must match the units of the expression.

```
limiter "AllConnections" 100 connections …
limiter "AllUploadBandwidth" 100 Mbps …
limiter "AverageUploadBandwidthPerFlow" 10 …
```

- Connection limiters must reference connection measurements, rate limiters must reference bitrate measurements, expression limiters must reference expression measurements.

```
limiter "AllConnections" 100 connections measurement "AllConnections"

measurement "AllUploadBandwidth" bitrate where source class "internal"
limiter "AllUploadBandwidth" 100 Mbps measurement "AllUploadBandwidth" …

measurement "AverageUploadBandwidthPerFlow" \
    expr(Measurement.AllUploadBandwidth / Measurement.AllConnections)
limiter " AverageUploadBandwidthPerFlow " 10 measurement \
    "AverageUploadBandwidthPerFlow" …
```

- For connection limiters, it is important to include a SandScript expression condition to exclude the flows that have been actioned from the measurement. For example, if the limiter is being used to control the tcp_reset action, the expression `expr(not(Flow.IsReset))` should be included.

```
limiter "GnutellaUploads" 100 connections where expr(Flow.ApplicationProtocol =
Protocol.Gnutella) \
    and transfer uni and sender class "internal" and \
    expr(not(Flow.IsReset))

limiter "GnutellaUploadBandwidth" 100 Mbps where expr(Flow.ApplicationProtocol =
Protocol.Gnutella) \
    and source class "internal" …
```

11.3 Optimizing Performance - Limit Distribution

The limit is a per-cluster limit. Since the traffic is not necessarily balanced to all modules evenly, the PTS adjusts the limit for each module according to its share of the load. For example, if one module gets twice as many connections as another, the limit on the first module will be twice that of the second.

Since traffic is constantly changing, the PTS continually evaluates the load on each module and re-adjusts the individual module limits accordingly. This consumes some system resources, and there are cases where it is not necessary.

Undivided Limit Distribution

Undivided limit distribution can be used when all limiter instances exist on exactly one module. In this case, setting the distribution parameter to 'none' can be used to indicate that the limit should not be divided amongst multiple modules, but that the full limit will be given to the single module on which the instance exists.

Using no distribution when an instance's traffic appears on multiple modules means that the output connections/bitrate are not respected.

Undistributed

Distribution can be optimized when load is evenly balanced to the different modules. In this case, the limit should be evenly distributed to the modules. This can be achieved by setting the distribution parameter to simple.

If it is known that the load is evenly balanced, each module should output $1/N^{th}$ the limit, where N is the number of modules. The simple distribution forces the number of connections/bitrate to be divided evenly. The load is not checked and no adjustments are made.

Note that the default behavior always produces the correct result: limits are distributed to accommodate the load, however, in some cases, the performance can be improved by using either of the two distribution options.

Using Limiters to Control Actions

Once you have declared a limiter, you can reference it (via a SandScript expression) in the condition of a rule.

Each limiter defines one or more SandScript fields (depending on the number of priorities defined). The SandScript fields return a boolean value, indicating whether or not the action should be applied. For example:

- For a connection limiter, returns true if the current number of active connections is above the specified limit, false otherwise.
- For a bitrate limiter, true/false depending on the output of the underlying bitrate measurement.

11.4 Examples Using Limiters

Example:
Example using unique by:

```
limiter "GnutellaUploadsPerIp" 10 connections where \
 expr(Flow.ApplicationProtocol = Protocol.Gnutella) and transfer uni and sender \
 class "internal" and expr(not(Flow.IsReset)) unique by sender-ip
limiter "GnutellaUploadsPerSubscriber" 10 connections where \
 expr(Flow.ApplicationProtocol = Protocol.Gnutella) and transfer uni and sender class \
 "internal" and expr(not(Flow.IsReset)) unique by (Subscriber)
limiter "GnutellaUploadsPerPolicyClass" 10 connections \
 where expr(Flow.ApplicationProtocol = Protocol.Gnutella) and transfer uni and \
 sender class "internal" and expr(not(Flow.IsReset))
 unique by (Internal.PolicyClass)
limiter "ConnectionPerServerIpAndServerPort" 10 connections\
 where server class "internal" and expr(not(Flow.IsReset)) \
 unique by (Flow.Server.IpAddress,Flow.Server.Layer4Port)
```

Example using priorities:

```
limiter "GnutellaUploadBandwidth" 100 Mbps where \
 expr(Flow.ApplicationProtocol = Protocol.Gnutella) and source class "internal" priority
"New" \
 priority "Existing"
limiter "GnutellaAndBittorrentUploadBandwidth" 100 Mbps where \
```

```
 expr(Flow.ApplicationProtocol("gnutella", "edonkey")) and \
 source class "internal" \
 priority "NewGnutella" priority "ExistingGnutella" \
 priority "NewBittorrent" priority "ExistingBittorrent"
```

Example using an expression:

```
limiter "GnutellaAndBittorrentUploads" 100 connections \
 expr(Measurement.GnutellaUploads + Measurement.BittorrentUploads)
```

This example defines a connection limiter to control the number of Gnutella uploads to the “external” network class:

```
limiter "GnutellaUploads" 10 connections where \
 expr(Flow.ApplicationProtocol = Protocol.Gnutella) and \
 transfer uni and receiver class "external" and \
 expr(not(Flow.IsReset))
if expr(Flow.ApplicationProtocol = Protocol.Gnutella) and transfer uni receiver class \
 "external" and expr(Limiter.GnutellaUploads.Limit) \
 then tcp_reset
```

This example limits by connection and allows two Gnutella uploads per subscriber:

```
limiter "GnutellaUploadsPerSub" 2 connections where \
 expr(Flow.ApplicationProtocol = Protocol.Gnutella) and transfer uni and receiver \
 class "external" and expr(not(Flow.IsReset)) \
 unique by subscriber
if expr(Flow.ApplicationProtocol = Protocol.Gnutella) and receiver class "external" and \

 expr(Limiter.GnutellaUploadsPerSub.Limit) then tcp_reset
```

This example limits the eDonkey upload rate to 100Mbps by managing unidirectional uploads and bidirectional flows. It also demonstrates the use of priorities:

```
if expr(Flow.ApplicationProtocol = Protocol.eDonkey) then reevaluate flow on
 DataTransferDirection
limiter "EdonkeyUpload" 100Mbps where expr(Flow.ApplicationProtocol = Protocol.eDonkey) \

 and source class "internal" \
 priority "New" priority "Existing"
```

12

DNS Monitoring

12.1 DNS QoE Monitor

Use the dns-qoe rule to monitor DNS server efficiency and mean time to respond (MTTR). These values can be used to provide a measure of subscribers' Internet experience.

The syntax is:

```
monitor-config rule-number {dns-qoe \
 {src-class all|all-internal|all-external|string} \
 {dst-class all|all-internal|all-external|string} \
 {enable on|off}}
```

Where:

src-class or dst-class Specifies which network class is examined for the analysis. src-class implies the network/policy class applies to the DNS clients and dst-class implies the network/policy class applies to the DNS server. Can be set to:

- all: all network classes. The default.
- all-internal: internal network classes only.
- all-external: external network classes only.
- string: a single network class from the *subnets.txt* file.

enable Set to on to enable the rule, off to disable it. Default is off.

For more information on the syntax of monitor-config rules, see *Using the Network Protection Module* on page 188.

Example:
This rule monitors the efficiency and mean time to respond for all requests from internal network classes to the dns-servers network class.

```
monitor-config 1 {dns-qoe src-class all-internal dst-class \
 dns-servers enable on}
```

12.2 DNS Top Domains or Users

Use the dns-top-domains rule to monitor the volume of requested domains and the dns-top-users rule to monitor the volume of requests executed by hosts.

The syntax is:

```
monitor-config rule-number {dns-top-domains|dns-top-users
 {class all|all-internal|all-external|string} {enable on|off}}
```

The filters that can be used are:

Filter	Description	Allowed	Default
class	Specifies which network class is examined for the analysis.	• all: all network classes • all-internal: internal network classes only • all-external: external network classes only • string: a single network class from the *subnets.txt* file	all

The parameters that apply are:

Parameter	Description	Allowed	Default
enable	Enables/disables the rule.	• on: enables the rule • off: disables the rule	off

Example:
This rule identifies the top domains being requested from all internal network classes.

```
monitor-config 1 {dns-top-domains class all-internal enable on}
```

Example:
This rule identifies which hosts from all internal network classes are executing the largest volume of requests.

```
monitor-config 1 {dns-top-users class all-internal enable on}
```

13

Load-balancing

13.1 Load-balancing

The load-balancer manages traffic between the CSPs' subscribers and external providers.

The function of the load-balancer is to optimally distribute this traffic among processing instances on a PTS cluster or a single PTS element, so the software can inspect and act on traffic flows according to SandScript policy. If a processing instance fails, traffic for that processing instance is shunted for no more than 5 minutes. If the processing instance is restored within that time, traffic continues to that processing instance. If the processing instance isn't restored, traffic is redistributed based on the load-balancing mode.

If a subscriber has recently been re-balanced from one module to another (for example, because a module was overloaded, or a PTS in the cluster was restarted), the subscriber information may appear for two separate modules until the subscriber record times out (20 minute maximum) on the original module.

The load-balancing modes are set using the `set config service load-balancer mode` CLI configuration command. The modes available are:

- `static` - balancing is achieved using 12 or 8 bits of the subscriber IP address (usually the least-significant bits).
- `ip-hash` - balancing is achieved using 8 bits of the subscriber IP address as for static mode, but the balancing decision is done centrally.
- `policy` - load-balancing policy, defined in policy.conf, groups the incoming traffic into bundles (based on IP address, subscriber ID, or subscriber attribute) and each bundle is assigned to a processing instance.

One PTS in a cluster is designated as the master load-balancer. If the master PTS element in a cluster is rebooted, most traffic is shunted until another master is elected and the load-balancing state is restored. To find out which PTS element is the master use the `show service load-balancer master` CLI command.

The configured load balancing mode must be the same on all the elements in a cluster. One way to ensure they match is to use central configuration. To verify that the configuration matches, look at the output of `show config service load-balancer` CLI command. In the case where the load-balancing is in SandScript policy, the relevant policy settings must match. To verify, look for `load_balance by` in the policy.conf file.

`static` Mode

This is the default mode, in which each element uses a deterministic algorithm to balance the IPs to the processing instances. If a processing instance fails, all traffic is redistributed based the new number of active processing instances. Balancing is achieved using 8 bits of the subscriber IP address on the PTS 14000 platform. The PTS 22000 and 24000 platforms use 12 bits by default, to compensate for the increased number of processing instances available in Sandvine's higher density PTS platforms. Without 12-bit balancing the overall efficiency of these platforms decreases in large clusters.

For IPv4 addresses, the least-significant bits are used by default. For IPv6 addresses, the default is the least-significant bits of the 64-bit prefix. Both static and IP-hash modes support IPv6 traffic.

Static load-balancing reduces variability between the processing instances and is very efficient. Unless you have a specific-use case, you should always use this mode on a PTS 24000 or 22000. However, in a deployment where one subscriber can have multiple IP addresses, or where policy is applied per-subscriber, this method can result in mis-applied policy.

`ip-hash` Mode

Balancing is achieved using 8 bits of the subscriber IP address, but the balancing decision is done centrally. This has the benefit that if one or more PTS element fails, only the traffic destined to that element is re-balanced.

Sandvine recommends this mode for PTS 14000 platforms.

`policy` Mode

Decisions are made centrally, so if any processing instance fails, only the traffic destined to that processing instance is re-balanced. Also, if one processing instance becomes over-loaded, and doesn't fail, traffic is dynamically off loaded to other processing instances with less load.

If a processing instance fails, traffic is redistributed based on the defined policy, such as least-load. A processing instance's load is redistributed once it exceeds 80%. When a processing instance is restored you could have a period of time where some processing instances have very heavy load and some processing instances have a very light load, but this will correct itself over time.

This mode gives the administrator the greatest control over the load-balancing algorithm. Use this mode to configure the load-balancer to respect geographical constraints in the cluster to reduce cluster-link traffic. However, due to the time it takes for subscriber traffic identification, this mode is not recommended for subscriber billing applications.

Certain SandScript features, some of which account for situations where subscribers are assigned more than one IP address, are only compatible with this mode, because all traffic for a subscriber must be balanced to the same processing instance for these features to work.

13.2 Centralized Load-balancing Configuration

To configure load-balancing to use central load-balancing or ip-hash, you must include a line in policy.conf that indicates how to aggregate IPs into a bundle (the bundle definition) and how to distribute the bundles (the distribution definition). A single bundle distribution definition is created for the cluster. If there is more than one definition, the last definition in the policy.conf file is acted upon.

The load-balancing syntax is:

```
load_balance by <bundle_definition> distribute by <distribution_definition>
```

Where `bundle_definition` is:

internal-ip	Balanced by internal IP. One IP per bundle.
subscriber	Balance by the subscriber. All IPs of a subscriber are assigned to a bundle.
(Subscriber.Attribute.<AttributeName>)	Balance by an attribute of a subscriber. All IPs with the same value of a defined attribute are grouped together in a bundle.
(Internal.Network)	Balance by network class. All IPs for a defined network class in subnets.txt are grouped together in a bundle.
(Internal.PolicyClass)	Balance by policy class. All IPs for a defined policy class in subnets.txt are grouped together in a bundle.
ip_hash	Balance by a hash of the internal IP. An 8bit hash of the IP according to the setup defined in the Network Configuration Guide under Advanced Configuration . An IP hash is equivalent to a bundle.

Where `distribution_definition` can be:

least_load	New bundles are assigned to the module with a "lower" load. This is the default configuration.
locality	IPs belong to a sub-cluster that the subscriber is connected to. Note that for this definition to be used, each element must be assigned to a sub-cluster.
cost cost_value limit limit_value	This distribution option can only be used with the subscriber attribute bundle definition.

Each new bundle is assigned a cost equal to the cost_value parameter. Whenever a bundle is assigned to a module, the cost of the bundle is charged to the module's limit. New bundles are always assigned to the module with the most available cost space.

cost_value The value assigned to each new bundle. When the bundle is assigned to a module, this value is charged to the module's limit.

limit_value The maximum cost that can be charged to each module.

cost limit limit_value {delim delimiter_token}

This distribution option can only be used with the subscriber attribute bundle definition.

Each new bundle is assigned a cost which is parsed from the subscriber attribute. Whenever a bundle is assigned to a module, the cost of the bundle is charged to the module's limit. New bundles are always assigned to the module with the most available cost space.

limit_value The maximum cost that can be charged to each module.

delimiter_token The delimiter token appearing in the subscriber attribute that separates the bundle value from the bundle cost. The default delimiter is (|).

13.2.1 IP Discovery

When load-balancing, each internal IP is mapped to a bundle and the bundle is assigned to a module.

When dealing with complex bundle definitions such as subscriber or attribute, each IP must individually be discovered and its bundling criteria evaluated before we can assign it to its final bundle/module. While this is happening several packets of this IP are shunted. After the assignment is made, no further packets are shunted and all traffic of the IP is inspected.

IpHash bundling maps IPs to bundle very fast and can be performed as packets ingress at line rate (assuming all hashes of IPs have been previously assigned to available modules). In this load-balancing mode all traffic is immediately bumped to its destination module. There is no loss of any packets in the flows.

13.2.2 Bundle Distribution

When executing the centralized load balancer, each internal IP that intersects the cluster (on either the subscriber or internet data intersect ports) is "discovered" and assigned to a bundle.

If the bundle has not been assigned to a module, a module of least load is selected and the bundle is assigned to it. All future discovered IPs that belong to the same bundle will be assigned to this module.

When clusters become large, it is desirable to split them into what is called sub-clusters. A sub-cluster is a method for logically partitioning the elements in a cluster into smaller collections of elements. Sub-clusters can have large geographical spans between them so it is important to limit the traffic that occurs over the links between them. If the bundle distribution for load-balancing is set to distribute by locality, sub-clusters need to be identified. Each element must be assigned to a sub-cluster.

Balancing by locality provides a method for distributing traffic that is sensitive to which sub-cluster the traffic for an IP is intersecting. When assigning an IP to a module, only modules in the sub-cluster that discovered the IP will be considered. If the sub-cluster is overloaded, then modules from another sub-cluster will be considered.

When balancing by locality, the load-balancing algorithm gives preference to assigning bundles to modules in the sub-cluster closest to the subscriber. In this mode, the “discovery” of IPs occurs on the subscriber port of the PTS. All bundles discovered on the subscriber port would preferentially be assigned to a module on the sub-cluster it intersected. This will result in a dramatic decrease in subscriber traffic on the inter sub-cluster links.

When the bundle definition is ip_hash, there is no notion of locality since IPs are not discovered in this mode. The bundles (that is, the ip hashes) are initially distributed uniformly amongst all available modules. In this load-balancing mode, locality is not

supported. When ip_hash bundles are redistributed in a cluster, they are assigned to modules of least load. This is the only valid bundle distribution for ip_hash. If locality is selected, it will fail to load.

13.3 Load-balancing by Locality

In central load-balancing mode, the PTS load-balances traffic evenly to all modules in the cluster, according to the load of the module. In the load-balancing policy, the distribution method can be set to locality, meaning that traffic will be balanced to a local module whenever possible. Elements are considered local if they are in the same sub-cluster.

Balancing by locality reduces the amount of traffic that flows over cluster links between sub-clusters. This is valuable, for example, when some elements in a cluster are geographically remote.

13.3.1 Load-balancing Failure Recovery

When policy specifies any distribution mode other than locality, the failure recovery mechanisms in the system only need to recover/replicate a minimal amount of information.

When the distribution mode is specified as locality, the failure recovery mechanisms need to recover/replicate a larger amount of information to make more accurate load-balancing decisions.

The failure recovery level is configurable through the `set config service load-balancer failure-recovery-level` CLI configuration command, and is set to low by default. When locality is specified as the distribution mode, the failure recovery level must be increased to medium (or higher) The failure recovery levels are:

- low failure recovery: master central load balancer replicates to all slave central load balancers. If a slave central load balancer is restarted its state is restored by the master central load balancer. IP assignments are only sent back to the originating load balancer.
- medium failure recovery: all recovery provided in low. IP assignments are broadcast out to all load balancers regardless of origin. This is the recommended failure recovery level.
- high failure recovery: all recovery provided in medium. If an load balancer is restarted, its state is restored by the master central load balancer. This level of recovery is only recommended for small clusters of three or four PTS elements due to the resources required for this level.

Consider a very simple cluster of two elements that is partitioned into two sub-clusters: SC1 and SC2. IPs that are discovered from traffic intersecting on the subscriber side of SC1 are assigned to modules in SC1 and vice versa. Traffic that is intersecting on the internet side of the sub-clusters is forwarded based on existing IP assignments and is not considered for new IP discovery. This is essentially where utilizing a failure recovery level of medium is required. If traffic is intersecting on the internet side of SC1 and subscriber IPs for flows in that traffic were initially discovered in SC2, we need to bump that traffic to the assigned module in SC2. This is essentially localizing the processing of traffic. If a failure recovery level of medium or higher is not used then the traffic intersecting on the internet side of a sub-cluster cannot be localized correctly. The broadcasting of IP assignments that is enabled in failure recovery levels of medium or higher, allows all elements in a cluster, regardless of sub-cluster, to localize traffic that is intersecting on the internet side. If a failure recovery level of medium or higher is not enabled, then traffic intersecting on the internet side of elements in different sub-clusters may be shunted instead of being bumped to the appropriate module.

This step must be taken when you modify the load-balancing policy to distribute by locality and before applying that policy. Then follow the standard procedure for updating the system after modifying the load-balancing policy. If this step is missed and applied to the system after the load-balancing policy has been applied, the state of the system must be manually cleared for locality to be fully utilized. Load-balancing by locality requires complete replication of load balancer state to all element in the cluster. The replication of state requires a separate configuration setting. This is necessary to correctly handle asymmetry of traffic in the network. Without this setting, asymmetrical flows intersecting internet ports on other elements in the cluster would not be inspected.

If the bundle definition is ip_hash, the amount of load-balancing state replicated to all elements is very small. Hence during any failure condition, any element can be elected the master load balancer and have full state recovery. The above settings for recovery will not matter.

13.4 Switching Load-balancing Modes

1. As an administrative user, edit `/usr/local/sandvine/etc/policy.conf` on each PTS.
   - If you are using IP hash mode or centralized load-balancing, add:

     ```
     load_balance by bundle_definition distribute by distribution_definition
     ```
   - If you using static load-balancing, remove or comment out any policy containing load_balance rules.
2. In the CLI, enter configure mode using the command: `configure`
3. To switch load-balancing modes run:

   ```
   PTS# set config service load-balancer mode <mode>
   ```

   Where <mode> is one of:
   - static—Load-balancing is achieved using 12 or 8 bits of the subscriber IP address (usually the least-significant bits).
   - ip-hash—Load-balancing is achieved using 8 bits of the subscriber IP address, as above, but the balancing decision is done centrally.
   - policy—This load-balancing policy is defined to group the incoming traffic into bundles (based on IP address, subscriber ID, or subscriber attribute) and each bundle is assigned to a module.
4. If load-balancing by locality:
   a. To create a sub-cluster, run the command:

      ```
      PTS# set config cluster sub-name <sub-name>
      ```

      where <sub-name> is the name of the sub-cluster.

   b. To set the failure recovery level, run the command:

      ```
      PTS# set config service load-balancer failure-recovery-level <level>
      ```

      where <level> is either low, medium or high.
5. After you have completed your configuration, commit the changes (the system will automatically reload, reboot or restart, which may impact service) with the `commit` command. This may take some time to complete.

13.5 Clearing the Master Load-balancer State

Some load-balancer configuration procedures require you to clear the state on the master load-balancer.

1. To identify the master load balancer, run the command: `show load-balancer master`
2. On the master load balancer, at a CLI prompt, run this command: `clear load-balancer state`

14

Captive Portal

14.1 Captive Portal Use Cases

These use cases provide examples of the application of the captive portal feature

14.1.1 Abuse

In this example the subscriber has a limit of 10 GB bandwidth allowance per week, and every 7 days the counter is reset. Purpose of the rule is to:

- warn the user when they reach 90% of their limit
- lock the user out when they reach 100% of their limit

The user is warned once every 60 minutes once they reach between 90% and 100% usage. When they reach 100%, they are directed to a specific URL. Note that the delay should be long enough to allow the entire frame in progress to complete download.

```
attribute "threshold_warning" type boolean
attribute "abuser" type boolean
if client "threshold_warning" = "true" and \
   expr(Flow.SessionProtocol = Protocol.Http) then \
    captive_portal
        "http://myportal.domain.com/abuserwarn.rvt" cycle 60min \
        interferences 1 delay 5sec
if client "abuser" = "true" and \
   expr(Flow.SessionProtocol = Protocol.Http) then \
    captive_portal "http://myportal.domain.com/abuser.rvt"
```

14.1.2 Pay as you go

In this example, the subscriber has purchased 60 minutes of Internet time at an airport facility. The purpose of the rule is to capture them at 60 minutes and give them an opportunity to purchase more Internet time.

```
attribute "pay" type boolean
if client "pay" = "true" and \
   expr(Flow.SessionProtocol = Protocol.Http) then \
    captive_portal "http://myportal.domain.com/paynow.rvt" \
        cycle 60min interference delay 20sec
```

14.1.3 Advertising-Funded Service

In this scenario, Internet service is an advertising funded "free" service. The service is free (or subsidized) but the subscriber is captured every 15 minutes and presented with some advertising.

```
attribute "freebie" type boolean

if client "freebie" = "true" and \
   expr(Flow.SessionProtocol = Protocol.Http) then \
    captive_portal "http://myportal.domain.com/advertisment.rvt" \
    cycle 60min interferences 3 delay 0min
```

In this use case Delay 0min means don't worry about inactivity. If the subscriber is inactive, capture them as soon as they do try something or if they happen to be inactive up to when the timer expires, they are locked out altogether.

14.1.4 URL Filtering

With this policy, specific sites can be blocked by responding to the client with a 404 error.

```
if server class "block" and \
   expr(Flow.SessionProtocol = Protocol.Http) then \
    captive_portal full_response "HTTP/1.1 404 Not Found\r\n\r\n"
```

This will send the 404 error to the client and reset the server.

14.1.5 Framed Experience

With the full_response keyword the policy author can add HTML/Javascript to send to the client in the captive portal packet.

```
# to avoid redirecting the client when they attempt to fetch a page from the
# CSP's website, the host field in the client's request is checked to ensure
# that it is not part of the "myisp.com" domain.
if (not(contains("myisp.com", Flow.Client.Stream.Http.Host)))\
and \
client "show_popup" = "yes" and not client "hide_popup" = "yes"\
then \
set_attribute client "hide_popup" = "yes" for 60sec \
and \
captive_portal full_response "HTTP/1.1 200
OK\r\nConnection: close\r\nContent-type: \
text/html;charset=utf-8\r\n\r\n \
<html><head><script>if(window.parent==window) \
{window.location=
'http://www.myisp.com/redirect.rvt?'+window.location.href;}\
else{window.location.reload();}</script></head></html>"

# attributes to control when to send the user the captive portal # response
attribute "hide_popup" values "yes"
local_attribute "show_popup" values "yes" "no"

# if popup frame is enabled and the user hasn't been sent the
# popup in 60 seconds
# and the protocol is HTTP, they are likely browsing the web
# then we send them a custom
# HTTP response with some html/javascript.
# the javascript will essentially redirect the browser to the
# content service provider's page with the original URL
# as an argument to the new URL. The CSP can then do whatever
# they need to in the response
if client "show_popup" = "yes" and not \
client "hide_popup" = "yes" and \
expr(Flow.SessionProtocol = Protocol.Http) then \
 set_attribute client "hide_popup" = "yes" for 60sec \
 and captive_portal full_response "HTTP/1.1 200
 OK\r\nConnection: close\r\nContent-type: \
 text/html;charset=utf-8\r\n\r\n \
 <html><head><script>if(window.parent==window) \
 {window.location=
 'http://www.myisp.com/redirect.rvt?'+window.location.href;}\
 else{window.location.reload();}</script></head></html>"
```

Note:
In the example above the last several lines are all part of one continuous string broken down into lines to fit it all on the page.

14.2 Captive Portal Overview

Use the captive portal action to direct a subscriber to a web page when specific conditions are identified.

For example, if Top Talkers is deployed, the captive portal can be used to inform subscribers of excessive bandwidth usage. If network protection is deployed, the captive portal can be used to redirect infected subscribers to a web page where they can access the tools necessary to clean their system.

The captive portal action is only used with a protocol http_get condition. It causes a temporary redirect message with the URL from the action to be sent to the HTTP client when a rule is matched.

Note:
Captive portal is also available in offline mode. Offline mode policy is exactly the same as inline policy.

The captive portal action has several parameters which indicate how often to redirect the subscriber and so forth.

Ensure that the redirected HTTP flow that results from the captive portal action is not itself redirected. This will prevent infinite redirections from occurring. This can be accomplished by checking that the flow isn't already requesting a page from your captive portal site or any host that your captive portal site references.

```
attribute "abuser" type boolean
define "IsCaptivePortalHost" = \
  Any(contains("captiveportal.myisp.com", Flow.Client.Stream.Http.Host), \
      contains("update.microsoft.com", Flow.Client.Stream.Http.Host), \
      contains("mcafee.com", Flow.Client.Stream.Http.Host))

PolicyGroup expr(All(Flow.SessionProtocol = Protocol.Http, \
                     Flow.Client.Stream.Http.Command = "GET",
                     IsCaptivePortalHost)) \
{
    if client "abuser" = "true" then captive_portal "http://myportal.domain.com/abuser.rvt"
}
```

15

Subscriber Behavioral Policy

15.1 Overview of Subscriber Behavioral Policy

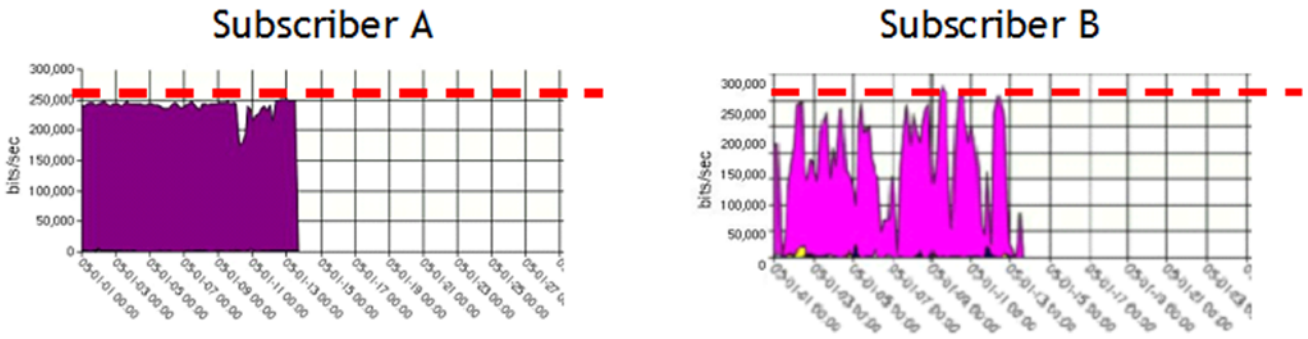
Subscriber Behavioral Policy applies policies to subscribers based on either their current or historical usage.

A service provider may choose to enforce a subscriber's service level or to target heavy users in congested situations to provide fair usage of shared resources. This feature is licensed separately from the standard policy functions of the Sandvine Policy Traffic Switch (PTS).

For example, the Sandvine system can be configured to identify subscribers who are top talkers; users who have transferred large amounts of data within a specified time period. Policies can then be applied to those particular subscribers.

In congested situations, however, what is more important to a service provider is the recent bandwidth usage of the subscriber. In most cases, it is desirable to apply a policy to a subscriber who has been transferring data at a consistently high rate for some time, rather than subscribers who are randomly bursting data.

In this diagram subscriber A exhibits sustained bandwidth use:



Subscribers can be classified according to their current usage (within last few hours), which is accomplished by using the Bandwidth Use Detection policy package. Then policies can be applied to the heavy users, for example, to those who are using peer to peer applications. Peer to peer applications are robust and will easily tolerate bandwidth reduction.

This feature includes a Bandwidth Use Management policy package, which defines complex shaping policies that include assigning traffic priorities based on protocols and one or more subscriber attribute values. By default, the bandwidth_use attribute defined by the Bandwidth Use Detection policy package is used. This package can be used to provide more subscriber friendly shaping policies to heavy P2P users, for example, that may cause local congestion for other subscribers within the Service Providers network.

Other modes include a mode related to session limiting. In this mode, the number of connections for peer to peer sessions can be limited based on subscriber attribute, such as the bandwidth_use attribute, as well as based on protocol.

15.2 SandScript Generators

SandScript generators are utilities that generate policy fragments that are then included in SandScript's policy file (policy.conf on the element or the centralized policy file for the cluster).

The available generators are:

- bandwidth_use_detection: tags subscribers based on their short-term bandwidth usage patterns
- bandwidth_use_management: shapes traffic fairly based on flow protocol and any number of subscriber attributes (by default, only bandwidth_use attribute set by bandwidth_use_detection policy).

Note:

Only one instance of the bandwidth_use_management package can be used in the policy.conf file. Multiple uses will cause the policy load to fail.

15.2.1 SandScript Generator Policy

SandScript policy generators can be created in the policy file on the element (/usr/local/sandvine/etc/policy.conf) or in the centralized policy file for the cluster.

The syntax is:

```
package bandwidth_use_detection|bandwidth_use_management {parameters}
```

Where:

package is a keyword that indicates this is a policy generator policy

parameters are package policy directives, similar to include policy directives with these differences:

- A number of named parameters may be specified
- The included policy file is dynamic (generated on the fly); the exact content depends on the provided parameter values.

Syntax of the package directive (in extended Backus-Naur form (EBNF)) is:

```
<package> ::= package <package_name> { <param_spec> { <param_spec> } }
<param_spec> ::= <param_name> {  = <param_value> }
<package_name> ::= <token>
<param_name> ::= <token>
<param_value> ::= <token>
```

Sandvine recommends you quote all parameter values (although the example below is valid).

Example:
For example:

```
package bandwidth_use_detection \
    protocols=p2p \
    max_download=10Mbytes \
    time_interval=60sec
```

15.3 Bandwidth Use Detection

The bandwidth_use_detection policy measures how much bandwidth a subscriber is using and then tags that subscriber one of the default bandwidth_use attribute values: low, medium, high, or abuse.

The syntax is:

```
package bandwidth_use_detection {parameters}
```

You must configure the time interval over which subscriber traffic is measured. For smooth operation use longer intervals, at least 15 minutes. Shorter intervals may cause frequent state changes. If sine shaping is applied, the time_interval must be twice as long as the sine shaping interval (see *Bandwidth Use Management Examples* on page 157).

The bandwidth_use_factor macro value (which drives the bandwidth_use attribute state machine) depends on the max_upload and max_download parameters. The formula is:

bandwidth_use_factor = (upload_bytes / max_upload) + (download_bytes / max_download)

Either max_upload or max_download must be specified. If one of them is missing, traffic will not be counted in that direction and will not contribute to the value of bandwidth_use_factor. For example, if only max_upload is specified, then the formula becomes:

```
bandwidth_use_factor = upload_bytes / max_upload
```

You can use weighted upload/download traffic to tag subscribers. For example, to specify that upload is more important than download.

The required parameters for bandwidth_use_detection are:

time_interval Time interval over which traffic is counted. Default is “300sec”. Recognized suffices are: sec, min, hour, and day; “sec” is assumed by default. For example: 1hour, 2hours, 10min, 120, 1day

max_upload Specifies total number of uploaded bytes over the specified time interval that is considered “too much”.

max_download Specifies total number of downloaded bytes over the specified time interval that is considered “too much”.

The optional parameters for bandwidth_use_detection are:

protocols A comma delimited list of protocols to aggregate. This list can include any of the protocol names that are recognized or a protocol group name. See the *Loadable Traffic Identification Package* document. For example:

```
protocols="bittorrent, voip"
```

attribute_name Used as the name of the subscriber attribute reflecting bandwidth usage. The default value is "bandwidth_use".

The output of this parameter feeds into the attributes function of the bandwidth_use_management parameter and must be the same if bandwidth_use_detection is selected as the input. The number of states for this parameter is not limited and the more states the more gradual the shaping. It is not recommended to use more than eight states. It is necessary to override the default to avoid name conflict if more than one instance of the policy generated by bandwidth_use_detection is included in the same policy file.

attribute_values A comma-delimited list of attribute values. The default value is "low,medium,high,abuse". The maximum length of an attribute value is 191 characters.

hysteresis Controls the transition from one state to the next. It defines the “stickiness” of higher attribute states. The value for this parameter is a percentage with a range of 0 — 70. The default value is 30.

local_attributes Set to "1" to use local attributes. Local attributes are not saved to database, which optimizes CPU and network load.

timeout Specifies the length of time before bandwidth_use_detection attribute_value expires. Default value is never expire.

For example, when bandwidth_use_detection policy sets subscriber as abuser, this abuser attribute will stay in the database until the timeout value is reached. At this moment the attribute will be reset as low. Note bandwidth_use_detection policy is still active and may modify this attribute_value based on policy rules. Resetting attribute_value may generate extra attribute state changes in the database. Therefore timeout value should be at least ten times longer than time_interval, to minimize number of state changes.

Protocol groupings that can be used in policy are:

Protocol Grouping	Policy Command	Protocols
All	*	All protocols (no filter)
Peer-to-Peer	p2p	BitTorrent, Gnutella, eDonkey, Ares, WinMX, Direct Connect, Gnutella 2, Manolito, Winny, Blubster
VoIP	voip or voice	H.323, MGCP, SIP, Skype

Protocol Grouping	Policy Command	Protocols
Gaming	games	PC Games, Quake, Lineage, Sony Online, Playstation2, Xbox
Web	web	HTTP, HTTP Proxy, SSL

15.3.1 Bandwidth Use Detection Results

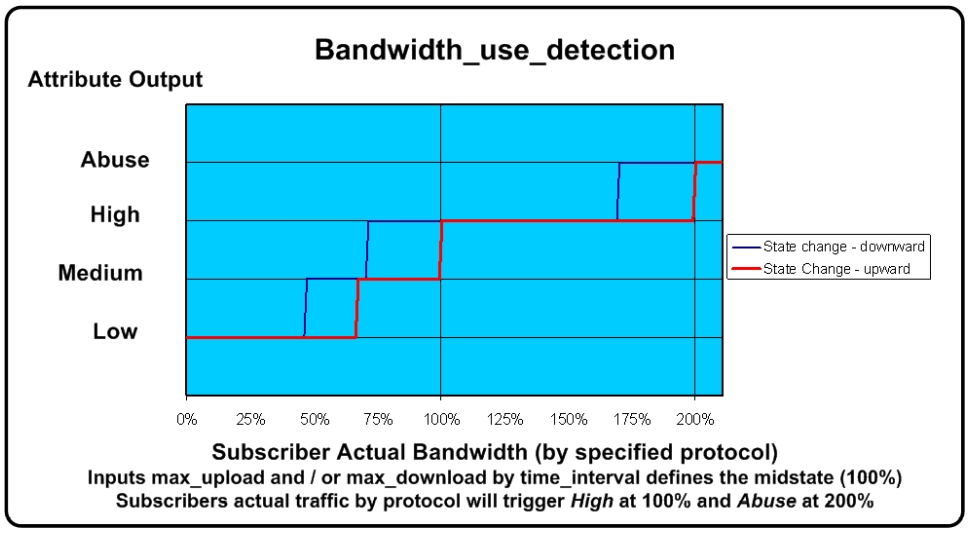
The bandwidth_use subscriber attribute is defined and set for some subscribers depending on their recent bandwidth usage.

The attribute may have these states: low, medium, high, abuse by default. This attribute is used by default by the bandwidth_use_detection policy described below.

Bandwidth_use_detection will determine an output of the policy generator based on one of the defined inputs:

- only max_upload
- only max_download
- combination of both max_upload and max_download

The value from one of these conditions is compared by the policy generator to the actual subscriber's traffic (by defined protocol) to determine the attribute value. If the actual traffic equals the stated value (from one of the inputs above) then the Policy input weight is 100%, which equals “High”. If the actual traffic is twice the stated values then the policy weight is 200% which equals “Abuse”. The default attribute_value is Low, Medium, High and Abuse.



Note the state of the policy generator does not lower to the next value until the actual traffic has dropped to the hysteresis value (default 30%).

15.3.2 Bandwidth Use Detection Examples

When creating an adaptive policy, the first line starts with the package keyword (for example, package bandwidth_use_detection …).

Example:

Count downstream P2P traffic only

```
package bandwidth_use_detection \
  max_download=1000000 \
  time_interval=60sec \
  protocols=p2p
```

This example sets the download traffic at 1,000,000 bytes over 60 seconds and only applies to P2P traffic, Upstream traffic is not included. If subscriber receives 1,000,000 bytes within 60 seconds the policy will rate the subscriber high. Subscriber will stay at the value high until either of these conditions are met: The total traffic exceeds 2,000,000Bytes in 60 seconds, then set to abuse. The total traffic drops below 576,000Bytes (hysteresis value) in 60 seconds, then set to medium.

Example:
An example for all protocols, a longer time interval, where both uploaded and downloaded bytes matter:

```
package bandwidth_use_detection \
  max_download=900000000 \
  max_upload=600000000 \
  time_interval=1hour
```

This invocation specifies high-water marks for both upload and download, so traffic in both directions is counted.

Example:
An example of two different generators operating at the same time:

```
#attribute_name=downstream_p2p
package bandwidth_use_detection \
   attribute_name="downstream_p2p" \
  max_download=1000000 \
  time_interval=60sec \
  protocols=p2p
#attribute_name=all_protocols
package bandwidth_use_detection \
 attribute_name="all_protocols" \
 max_download=9000000 \
 max_upload=600000000 \
 time_interval=1hour
```

15.4 Bandwidth Use Management

Bandwidth use management implements a policy for different types of shaping or session limiting based on:

- subscribers
- destination networks
- values of zero or more subscriber attributes. For example, bandwidth_use attribute set by attribute_detection policy (this one is used by default) or any other attributes (such as representing service tier).
- flow protocols (optional)

Total flow priority is calculated by adding priorities of all contributing criteria (subscriber attribute values and/or protocols). Combined flow priorities are translated into shaper queue weights, shaper bandwidths or connection limits, depending upon the mode.

You can decide which criterion is more important (has a higher priority). For example, values of attribute "a" may have weights 20, 40, 60 (most important), values of attribute "b" weights 5, 10, 15 (less important), and protocols weights 1, 2, 3 (least important).

The total output of each policy must be a value between 0 and 100.

Depending on specified priorities, criteria dimensions may either overlap ("good" protocols of "silver" users may have higher priorities then "bad" protocols of "gold" users), or be completely isolated (any protocol of a "gold" user is more important than any protocol of a "silver" user).

The policy package has several different modes, which implement different types of shaping or session limiting. These modes are:

min_upload_bandwidth The minimum upload bandwidth.

Only used when mode is set to "fair", "fixed", or "simple". If bandwidth parameters are not specified for a direction, then traffic in that direction is not shaped.

max_upload_bandwidth The maximum upload bandwidth.

Only used when mode is set to "fair", "fixed", or "simple". If bandwidth parameters are not specified for a direction, then traffic in that direction is not shaped.

min_download_bandwidth The minimum download bandwidth.

Only used when mode is set to "fair", "fixed", or "simple". If bandwidth parameters are not specified for a direction, then traffic in that direction is not shaped.

max_download_bandwidth The maximum download bandwidth.

Only used when mode is set to "fair", "fixed", or "simple". If bandwidth parameters are not specified for a direction, then traffic in that direction is not shaped.

mode Sets the mode for bandwidth use management. Possible values are:

- weighted - a single shaper, with a number of queues with different priorities. Flows are shaped through one of the queues depending on combined flow priority.
- fair - a number of shapers with different bandwidths. Flows are shaped by IP through one of the shapers depending on combined flow priority (equitable sharing).
- fixed - a number of shapers with different bandwidths. Flows are shaped unique by IP through one of the shapers depending on combined flow priority (fixed per subscriber bandwidth limits).
- simple - a number of shapers with different bandwidths. Flows are shaped through one of the shapers depending on combined flow priority (fixed total limits).
- limiting - this mode does not use shaping at all, so it may be used in offline mode. It limits connections using per-subscriber connection counters (measurements). The minimum number of connections when the flow is killed depends on combined flow priority; lower priority flows are killed first.

  In fair, fixed, and simple modes number_of_shapers shapers are created with bandwidths ranging from (min_*_bandwidth to max_*_bandwidth). In weighted mode, a single shaper has number_of_queues queues with weights ranging from min_queue_weight to max_queue_weight.

upload_bandwidth Total upload bandwidth of the shapers.

You must specify at least one of upload_bandwidth or download_bandwidth parameters if mode equals weighted. There is no default, so if one of the parameters is not specified, traffic in that direction is not shaped.

download_bandwidth Total download bandwidth of the shapers.

You must specify at least one of upload_bandwidth or download_bandwidth parameters if mode equals weighted. There is no default, so if one of the parameters is not specified, traffic in that direction is not shaped.

protocols Specifies protocol priorities. This parameter takes a comma-delimited list of protocol=priority pairs. Protocol names are the same strings as those used in policy protocol conditions. Priorities must be in 0 to 100 range. Higher numbers correspond to higher priorities.

The protocols definition must be on the same line. Example:

```
protocols=”bittorrent=1,ftp=2,http=3,*=4”
```

A special "*" catchall value is recognized. It means "all other protocols". If "*" is not set, only explicitly listed protocols will be managed, the rest will be unmanaged. Also these special protocol groups are recognized for convenience: p2p, web, voip. The default parameter value is "p2p=1,web=3,voip=4,*=2"

attributes

Specifies names of subscriber attributes, their values, and base priorities of each value. Any number of attributes may be used for traffic prioritization.

Attribute specifications have the syntax:

```
name: value=priority, value=priority; name: value=priority, value=priority
```

For example, this attributes parameter sets two subscriber attributes, service_level and bandwidth_use:

```
attributes=”service_level: silver=5, gold=10, platinum=15; bandwidth_use:
 *=7, moderate=5, high=2, abuse=0”
```

The formal Extended Backus-Naur form (EBNF) syntax is:

```
<attributes> ::= <attr> { ; <attr> }
<attr> ::= {<name> : <valprio> { , <valprio> }}
<valprio> ::= {<value> = <prio>}
<value> ::= <token> | *
<prio> ::= <integer>
```

The default value of the attributes parameter specifies some reasonable priorities for bandwidth_use attribute only. If the bandwidth_use_detection policy is not used or uses non-default values, the default must be overridden. The default value is: "bandwidth_use: abuse=0, high=1, medium=2, *=3". It is recommended to specify this parameter explicitly as the default is subject to change.

If bandwidth_use_detection policy uses other values, this parameter must also be overridden and mention the same values.

An empty string may be specified, in which case no subscriber attributes will be used for traffic prioritization.

The special attribute value "*" means "all other values" (including no value). If "*" is not mentioned, only flows of subscribers with listed values of the attribute will be managed.

Any subscriber attribute may be used for traffic prioritization, not only bandwidth_use set by the bandwidth_use_detection policy, for example, service tier, an attribute set by a top talkers rule, etc. It's the responsibility of the policy writer to define those attributes and write rules setting them. Using the bandwidth_use_detection policy is optional. For example, this policy package can just use protocols to determine flow priority.

subscriber_network_classes

Specifies priorities of network classes on the subscriber side of the flow. A comma-delimited list of name=priority pairs, where name denotes a network class name. The name definition must be on the same line. An asterisk (*) means "all other network classes". If an asterisk is not indicated, only the explicitly listed network classes (on the subscriber side of the flow) will be managed. In this example, all flows involving subscribers in the internal network class will get a small priority boost (+1).

```
subscriber_network_classes="internal=1, *=0"
```

Empty by default.

other_network_classes

Specifies priorities of network classes on the peer side of the flow. A comma-delimited list of name=priority pairs, where name denotes a network class name. This is the same as the subscriber_network_classes parameter but the other side of the traffic flow. The name definition must be on the same line. An asterisk (*) means "all other network classes". If an asterisk is not indicated, only the explicitly listed network classes (on the peer side of the flow) will be managed.

In this example, all flows involving peers in the internal network class will get a small priority boost (+1).

```
other_network_classes=”internal=1, *=0”
```

Empty by default.

condition A guard condition applied to the entire shaper. Shaping will happen only when the condition is true. May be any policy language expression that can be used as a PolicyGroup condition, for example, "time hours 1700 2300". The default is empty so shaping is unconditional. Note that if the condition argument has double quotes in it (for example, if there are attributes) the policy will not compile. You need to use single quotes.

download_queue_size The size of the download queue.

Specified in bytes, slots, or ms. Empty (unspecified) by default.

upload_queue_size The size of the upload queue.

Specified in bytes, slots, or ms. Empty (unspecified) by default.

queue_size Sets both upstream and downstream queue sizes to the same value.

Specified in bytes, slots, or ms. Empty (unspecified) by default.

shaping_mode Used to determine shaper queue weight range. Can be one of:

- aggressive - the highest queue weight is nine times higher than the lowest weight
- moderate (the default)
- mild - the highest weight is only twice as high as the lowest weight

For finer-grained control over queue weight range (shaping aggressiveness) use min_queue_weight and max_queue_weight parameters.

min_queue_weight The minimum combined priority of the least important traffic will be sent to the shaping queue with this weight as the scheduling priority.

Everything in between this an max_queue_weight will be proportionally mapped to this range (number_of_queues controls the precision of the mapping). Queue weights must be in the range of 1 to 100. When specified, shaping_mode parameter (if provided) is ignored. The default value is 20 (which corresponds to shaping_mode=moderate).

max_queue_weight The maximum combined priority of the most important traffic will be sent to the shaping queue with this weight as the scheduling priority.

Everything in between this an min_queue_weight will be proportionally mapped to this range (number_of_queues controls the precision of the mapping). Queue weights must be in the range of 1 to 100. When specified, shaping_mode parameter (if provided) is ignored. The default value is 90 (which corresponds to shaping_mode=moderate).

number_of_queues The number of shaper queues. At least 4, no more than 20. The default value is 9. Affects prioritization resolution only (how fine-grained the distinction is). This parameter is used only when mode="weighted".

number_of_shapers The number of shaper queues. At least 1 and no more than 20. The default value is 3. This parameter is only used when mode="fair" or "fixed" or "simple".

timeout Indicates when the attribute set by the policy will expire. If not set, the policy will never expire. This means that when a subscriber goes offline and is unmapped, the database will keep the last value. Do not use with local attributes.

Notes on subscriber_network_classes and other_network_classes

The rules for matching subscriber_network_classes and other_network_classes are:

- If client side is a subscriber then client IP is matched against subscriber_network_classes and server IP is matched against other_network_classes.

  Else if server IP is a subscriber then server IP is matched against subscriber_network_classes and client IP is matched against other_network_classes

In this example, connections to other internal nodes have the highest priority (3). Connections to the good_external network class have a lower priority. Connections to the bad_external network class have the lowest priority. Everything else has a priority of 1.

```
other_network_classes="internal=3, good_external=2, bad_external=0, *=1"
```

Notes on shapers and impacts from protocols and attributes

Protocols and attributes are prioritization dimensions. They contribute to the combined priority independently (are added). Every dimension is equal (protocols, attribute, another attribute, network, and so on). How important a dimension is depends on the assigned priority range. For example:

```
protocol priorities are in 0..10 range
attribute a priorities are in 5..15 ranges
attribute b priorities are in 1..3 range
So the total priority range is 6..28.
```

Every concrete combination ((protocol, attr_a, attr_b) tuple) will have certain combined priority in this range. For example:

```
protocol "p" has priority 2.
attribute a value (for example, "a1") has priority 7.
attribute b value (for example, "b2") has priority 2.
Combined priority of ("p","a1","b2") tuple is 11.
Or, if you normalize it in 6..28 range: (11 - 6) / (28 - 6) * 100% = 23%
```

So far it's absolutely the same for all management modes. The difference is only in how priorities are enforced:

- Flows with combined priority of 0% will get the worst bandwidth, shaper queue priority, connection limit, and so on (depending on mitigation mode).
- Flows with combined priority of 100% will get the best bandwidth, shaper queue priority, connection limit, and so on (depending on mitigation mode).
- Flows with combined priority of 50% will get something in between.

The generated policy contains nested attribute checks using PolicyGroups with conditions. The nesting takes place in the order of the attribute declaration. The values are also checked in the order of declaration, except for "*", which is always checked last.

15.4.1 Bandwidth Use Management Examples

Example:
Default policy

```
package bandwidth_use_management download_bandwidth=200Mbps upload_bandwidth=100Mbps
```

The policy will use flow protocols and a single subscriber attribute: bandwidth_use (which means policy generated by bandwidth_use_detection has to be included first).

Default, shape upstream only, and only during peak hours

```
package bandwidth_use_management upload_bandwidth=100Mbps condition="time hours 1700-2300"
```

Downstream traffic is not shaped (download_bandwidth was not specified). Upstream shaping takes place only between 5pm and 11pm.

Example:
Shape based on protocol priorities only. Don't use any attributes

```
package bandwidth_use_management download_bandwidth=200Mbps upload_bandwidth=100Mbps \
attributes="" \
protocols="p2p=0,voip=4,web=3"
```

Note that it is necessary to override the default value of the attributes parameter, since its default value uses the bandwidth_use attribute. This way there is no need to include bandwidth_use_detection policy.

This example also explicitly specifies protocol priorities. Only mentioned protocol groups will be shaped (there is no "*" item) for all subscribers.

Example:
Shape based on bandwidth_use only. Don't check protocols

```
package bandwidth_use_management download_bandwidth=200Mbps \ upload_bandwidth=100Mbps \
protocols=""
```

The default value of attributes uses bandwidth_use. The default value of protocols was overridden with an empty string, so the generated policy won't check protocols.

This example is using the default mode of "Weighted" for the shaper, with all subscribers being shaped to 200Mbps download and 100Mbps upload. The weighting on all subscribers will depend on their bandwidth_use_detection value. "low" subscribers will have a much higher weighting then "abuse". Therefore "low" subscribers will have a greater opportunity at the output bandwidth of the shaper than "abuse".

Example:
Shape only abusers and high users, and only P2P protocols

```
package bandwidth_use_management upload_bandwidth=100Mbps \ download_bandwidth=200Mbps \
protocols="p2p=0" \
attributes="bandwidth_use:abuse=0,high=1"
```

This avoids shaping overhead for most of traffic. Only P2P traffic is shaped, and only "bad" users are affected. There are only two priority levels for the shaped traffic.

Example:
Use another attribute: tier=silver/gold/platinum

```
package bandwidth_use_management download_bandwidth=200Mbps \ upload_bandwidth=100Mbps \
attributes="bandwidth_use: *=3,moderate=2,high=1,abuse=0;
tier: silver=3,gold=6,*=9;"
```

Note that the attribute definition must be on one line. In this example, it wraps. The last line in the definition is actually a continuation of the previous line.

This policy uses two attributes: bandwidth_use and tier, and default protocol priorities. All flows will be shaped even if there are other (not explicitly mentioned) attribute values or some subscribers don't have any of the two attributes (both attribute specifications contain "*").

Example:
Exact protocols, no attributes

```
package bandwidth_use_management download_bandwidth=200Mbps \ upload_bandwidth=100Mbps \
attributes= \
protocols="bittorrent=0,edonkey=1,ares=2,http=5,ftp=6,smtp=4, \
pop3=4,*=3"
```

Exact protocols, no attributes but don't shape other protocols

```
package bandwidth_use_management download_bandwidth=200Mbps \ upload_bandwidth=100Mbps \
attributes="" \
protocols="bittorrent=0,edonkey=1,ares=2,http=5,ftp=6,smtp=4,\
pop3=4"
```

The only difference is that protocol "*" is missing, so other (unlisted) protocols won't be shaped at all.

Example:
Two attributes: bandwidth_use and tier.

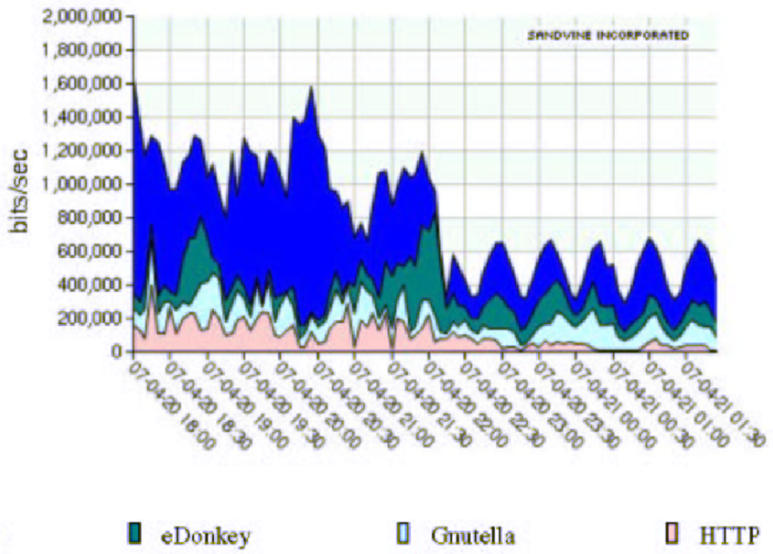
```
package bandwidth_use_management download_bandwidth=200Mbps \ upload_bandwidth=100Mbps \
mode=aggressive \
attributes="tier:silver=3,gold=6,*=8;bandwidth_use:*=3,moderate=2,high=1,abuse=0"
```

Using attributes and time

```
attribute "download_abuser" values "true" "false"
package bandwidth_use_detection \
    max_download=400000000 \
    time_interval=1hour
# using single-quotes for the attributes and using 0000 for 12mid-night.
package bandwidth_use_management \
    condition="(client 'download_abuser'='true' or server 'download_abuser'='true') and
time hours 1400-0000 wdays mon-fri" \
    download_bandwidth=10Mbps
```

Example:
Sine wave shaping to customer data flow

The results of using this example policy are shown in this report for bandwidth by protocol (receive):




```
package bandwidth_use_detection  max_upload='8000000' max_download='20000000'
time_interval='4800sec'  protocols='*' attribute_name='bandwidth_use'
attribute_values='low,medium,high,abuse'  local_attributes='0'
# here are the local attribute needed to drive the sine wave
local_attribute "sine" values "s0" "s1" "s2" "s3" "s4" "s5" "s6" "s7"
local_attribute "sine_timer" values "set"

PolicyGroup not "sine_timer" = any {
        if "sine" = "s0" then \
        set_attribute "sine" = "s1" and \
        set_attribute "sine_timer" = "set" for 300 sec

        if "sine" = "s1" then \
        set_attribute "sine" = "s2" and \
        set_attribute "sine_timer" = "set" for 300 sec

        if "sine" = "s2" then \
        set_attribute "sine" = "s3" and \
        set_attribute "sine_timer" = "set" for 300 sec
```

```
        if "sine" = "s3" then \
        set_attribute "sine" = "s4" and \
        set_attribute "sine_timer" = "set" for 300 sec

        if "sine" = "s4" then \
        set_attribute "sine" = "s5" and \
        set_attribute "sine_timer" = "set" for 300 sec

        if "sine" = "s5" then \
        set_attribute "sine" = "s6" and \
        set_attribute "sine_timer" = "set" for 300 sec

        if "sine" = "s6" then \
        set_attribute "sine" = "s7" and \
        set_attribute "sine_timer" = "set" for 300 sec

        if true then \
        set_attribute "sine" = "s0" and \
        set_attribute "sine_timer" = "set" for 300 sec
}

# now this is the shaping policy
# only abuse and high users are shaped
# mode is fixed shaping, meaning each ip got into a specific shaper,
# not shared with other ip
# there are 10 shapers, min 0.3Mbps and max 1.0Mbps
# but the actual bandwidth will dance between 0.3 and 0.6
# because anything more than 0.5 will be treated as abuser and shaped
# down

package bandwidth_use_management mode=fixed'\
number_of_shapers=10 \
attributes='bandwidth_use:high=8,abuse=0;sine:s0=0,s1=2,s2=4,s3=6,s4=8,s5=6,s6=4,s7=2' \
protocols=* \
min_upload_bandwidth=300Kbps \
max_upload_bandwidth=1Mbps \
min_download_bandwidth=300Kbps\
max_download_bandwidth=1Mbps
```

15.4.2 Bandwidth Use Management in Limiting Mode

The bandwidth use management policy package can be used in offline deployments with the "limiting" mode. In this mode, this package employs the same traffic prioritization principles described in the previous section, but using session limiting as the action, in particular, the "unique by subscriber" limiting capabilities. By default, only P2P protocols are examined when in the "limiting" mode.

The limiting mode can also be used when the PTS is deployed inline; in this case, an additional action of "block" is also supported.

The more important connections will get higher connection limits depending on the prioritization criteria described above. Only the number of connections can be limited at this time (not bandwidth).

The parameters specific to `bandwidth_use_management mode=limiting` are:

min_connections Minimum allowed connections for the specified protocols. Default is 0. The actual limit is calculated based on prioritization criteria (attribute values and/or protocols) in the same manner as for shaping.

max_connections Maximum allowed connections for the specified protocols. Default is 20. The actual limit is calculated based on prioritization criteria (attribute values and/or protocols) in the same manner as for shaping.

action Specifies the action to take to decrease the number of connections. The default is "tcp_reset".

udp_action

Specifies an action to throttle down UDP flows. If specified, the generated policy checks the IP protocol, and issues the specified action if the protocol is UDP. The default value is "" (no special action for UDP).

For example:

```
udp_action=”block” action=""
```

Only block UDP flows until either connection count or bandwidth use drops to an acceptable level. Leave other types of flows alone.

Note:
At least one of action or udp_action parameters must have a non-empty value.

measurement_name

Name of the connection-counting measurement. Optional. Override the default if more than one fair_weighted_shaping policies (in limiting mode) are included in the same policy file.

These parameters are ignored in limiting mode:

- download_bandwidth
- upload_bandwidth
- download_queue_size
- upload_queue_size
- number_of_queues
- shaping_mode
- shaper_base_name

15.4.2.1 Bandwidth Use Management in Limiting Mode Examples

Example:
Default

```
bandwidth_use_management mode=limiting min_connections=2 \ max_connections=30
```

Uses bandwidth_use attribute (by default) to determine the maximum number of allowed P2P connections. Abusers get only two connections. Note that protocols="p2p" by default in limiting mode.

Example:
Different priorities for a few concrete P2P protocols

```
bandwidth_use_management mode=limiting min_connections=2 \ max_connections=30 \
protocols="edonkey=0,p2p=2"
```

The same as the default, but eDonkey will be limited first (the lowest priority).

Example:
Use an additional attribute for prioritization

```
bandwidth_use_management mode=limiting min_connections=2 max_connections=30 \
attributes="bandwidth_use:low=5,medium=4,high=3,abuse=0; tier:gold=4,*=2" \
protocols="edonkey=0,p2p=2"
```

Overrides attributes to use an additional attribute: tier. Note, that according to the specified priorities, high users of the gold tier will have the same priority as low users of all other tiers (3+4=5+2) for the same protocol.

Example:
Don't use attributes at all, just protocols

```
bandwidth_use_management mode=limiting min_connections=2 max_connections=30 \
attributes="" \
protocols="edonkey=0,gnutella=1,bittorrent=2" \
udp_action="block"
```

The default value of attributes is overridden with an empty string (which means bandwidth_use_detection policy is not needed). eDonkey will be limited first (the lowest priority), and so on. UDP sessions of the tracked protocols will be blocked. TCP sessions will be reset (the default).

16

Destination

16.1 Destination Groups

A destination group can be used to balance teed or diverted traffic across a group of tee/divert destinations.

The group members can be optionally health checked to automatically shift load away in the case of a failure.

The syntax for a destination group is:

```
destination "<name>" group destinations "<dest_name1>" {"<dest_name2>" …}
    {healthchecks "<healthcheck1>" {"<healthcheck2>" …}}
    {ramp_interval <n>} {ramp_iterations <n>}
```

Where:

name	The name of the destination group.
dest_name	The name of a previously declared destination. The type of the destinations within the group must be consistent (either all tee, all divert, or all divert_sequence). Traffic is balanced evenly across the members of the group. The balancing is done on the internal IP address of each flow, ensuring that all traffic for all internal hosts is always balanced to the same destination.
healthcheck	The name of a previously declared healthcheck. The list of healthcheck names specifies which healthchecks get applied to the members of the group. If specified, traffic is balanced across the group members for which ALL healthchecks are passing. See *Health Check Configuration* on page 165 for more information.
ramp_interval	Specifies the duration of each “ramp-up” iteration. Units are seconds and minutes.
ramp_iterations	Specifies the number of ramp-up iterations to perform before teeing/diverting a full load of traffic to the destination.

Restrictions:

- Groups of groups are not permitted.
- All destinations within the group must be the same type (all divert or all tee).

Traffic can be slowly applied to destinations within a group by specifying the ramp_interval and ramp_iterations options. If so specified, traffic that is teed or diverted to each destination will ramp up on startup and each time the destination transitions from down to up.

For example, if the ramp interval is set to one minute and the ramp iterations is set to five, traffic is ramped up to the device over five minutes. During the first minute that the divert host is up, one fifth of the flows will be sent to it; during the second minute, two fifths of the flows will be diverted and so forth. After five minutes have elapsed, all of the flows will be sent to the divert host. This prevents the divert host from being overloaded and allows the host to perform the necessary tasks the process the flows that it receives.

16.2 Health Check Configuration

A health check enables verification of whether a host is "up" to determine if traffic should be sent to the host.

If a health check is not conducted, traffic is sent to all hosts in the group, irrespective of whether a host is available or not. Health check can be used for both tee and divert. Health checks are used in combination with destination groups to monitor destinations and automatically shift traffic away if the destination becomes faulted. The types of health checks supported are: ping, http, and inline.

Ping Health Check Syntax

A ping health check sends an ICMP echo request. The device is up if an echo response is received within the specified amount of time. The interval and timeout options are required. The syntax is:

```
healthcheck "<name>" ping
 interval <n>
 timeout <n>
 {retry <n>}
 {retry_failure <n>}
```

HTTP Health Check Syntax

An HTTP health check sends an HTTP request. HTTP health checking allows the health of a host to be determined using an HTTP HEAD request to an HTTP server that it is running. The response must be received within the specified time and the response must match a regular expression, which by default is the “200” (OK) code. Some aspects of the request and port numbers are configurable. The interval and timeout options are required. The syntax is:

```
healthcheck "<name>" http
 interval <n>
 timeout <n>
 {retry <n>}
 {retry_failure <n>}
 path "/request-file.html"
 {response_regex "regex"}
 {http_version "version"}
 {port <n>}
```

Inline Health Check Syntax

An inline health check sends an IP packet over the service plane and relies on layer 2 switching to route the packet back to the service plane. As such, inline health checks can only be used with full divert sequence destinations. The host sequence is up if the packet is received by the PTS (within a configurable amount of time).

```
healthcheck "<name>" inline
 interval <n>
 timeout <n>
 {retry <n>}
 {retry_failure <n>}
 {src_addr ipaddr}
 {dst_addr ipaddr}
 {ttl number}
```

Health Check Options

The health check options are:

name — Name of the health check.

interval — How frequently to perform the health check. Units are ms, seconds and minutes and are mandatory. Only ms can be abbreviated and all units are plural. For example 5seconds.

timeout — The amount of time to wait for the health check response. If the response has not arrived by the time out, the health check times out and fails. timeout must be less than the interval option. Units are ms, seconds and minutes and are mandatory. Only ms can be abbreviated and all units are plural. For example 20ms.

retry — If the host is marked up, the number of times the health check is retried after the initial failure before the host is marked down. For example, if this value is set to 2, there must be 3 consecutive failures before the host is marked down.

Default is 0.

retry_failure — If the host is marked down, the number of times the health check is successful before the host is marked up. For example, if set to 1, there must be 2 consecutive passes before the host is marked up.

	Default is 0.
src_addr	The source IP address to use in the IP header of the health check packet.
	Private IP address 10.0.0.0.
dst_addr	The destination IP address to use in the IP header of the health check packet.
	Private IP address 10.0.0.1.
ttl	The time to live value to use in the IP header of the health check packet.
	Default is 1.
path	Resource to request from the server in the "HEAD" line.
response_regex	Regex to match. If the response matches this regex, the health check passes, otherwise it fails.
	Default is "\AHTTP/1.[01][\t]+200[\t]". The default response_regex ("\AHTTP/1.[01][\t]+200[\t]".) expects HTTP:/1.1 followed by white space followed by 200 followed by white space or HTTP:/1.0 followed by white space followed by 200 followed by white space. The \A in the regex means that the signature must be anchored at packet start.
http_version	The HTTP version to be used in the request. Supported versions are 1.1 and 1.0.
	Default is 1.1.
port	The TCP port number of the HTTP server.
	Default is 80.

Health Check Example

In this example the HTTP health checker generates a request to the hosts, with the host's IP, version and path filled in. The User-Agent is fixed to the indicated value. HTTP can use this agent to determine if it is being health checked.

```
healthcheck "HTTP_check_1" http \
    interval 10 seconds \
    timeout 800 ms \
    path "/index.html" \
    http_version "1.1" \
    retry 1 \
    retry_failure 4 \
    response_regex "\AHTTP/1.1 202"

destination "host1" divert ip 1.0.0.2
    destination "group1" group destinations "host1" \
    healthchecks "HTTP_check_1"
```

The example initiates a check every ten seconds this request to address 1.0.0.2 on port 80 (the configured IP address of the destination being checked).

```
HEAD /index.html HTTP/1.1
Host: 1.0.0.2
User-Agent: PTS Health Check/1.0
Connection: close
```

A response similar to this must be received within 800 ms of the request:

```
HTTP/1.1 202
```

16.3 Teeing and Capturing to a File

16.3.1 Overview of Tee

The tee feature redirects a copy of the traffic flow to a file or to another device for further processing.

It enables mirroring of traffic for lawful intercept regulations or for improved network monitoring. To tee you must :

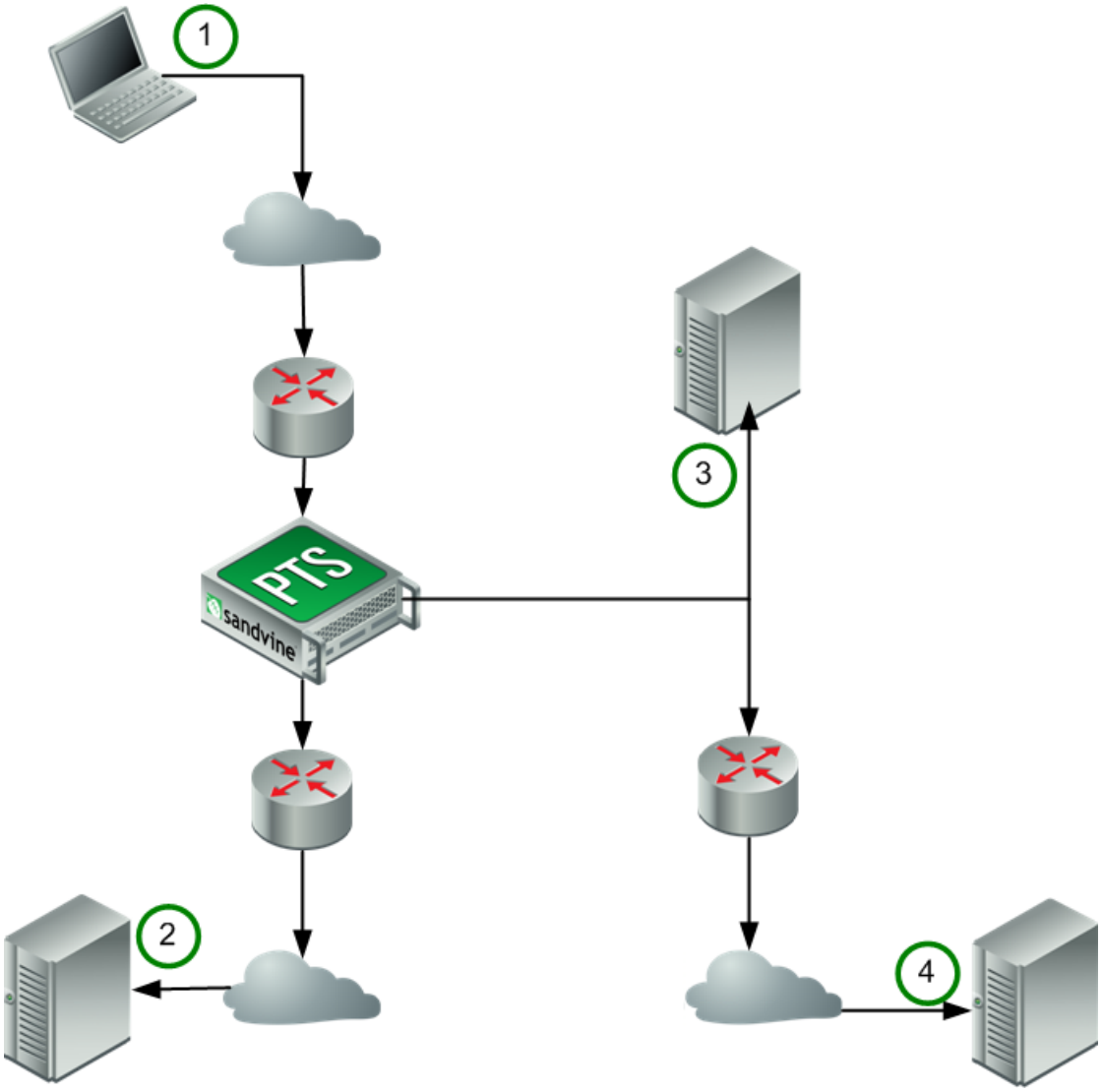
1. Create a configuration that identifies external equipment as the destination for the traffic
2. Create a tee action to tee packets to the destination.

Note that tee is licensable.

16.3.1.1 Teeing Packets

The tee action enables packet by packet copying with optional encapsulation.

You can send the copied packets to up to four specified destinations. You can tee packets to two destinations. In this example:



1. Traffic comes from the subscriber to the PTS.
2. The PTS sends the original packets to the original destination.
3. Traffic is copied and sent to a data base on the same layer 2 network as the PTS.
4. A second copy is sent to a database that is multiple hops away on a different layer 2 network.

Teed traffic can be balanced across a group of destinations by specifying a destination group.

- Tee supports a maximum of 4096 individual headers.
- Each direction of a flow may be teed to four destinations, including both tee and file destinations. Additionally, one divert destination may be selected per flow.
- Sandvine recommends that teeing be done out of the service port. While teeing out of the control port is possible, this should be limited to no more that 10% of the interface rate. If the rate is any higher, the control port may become unresponsive to remote terminal sessions via SSH. The element is still manageable via the serial port.

16.3.2 Destinations

The types of destinations for teed traffic that can be defined are:

- tee - a copy of a packet is sent to another host
- divert - the flow is redirected to a different host.
- divert_sequence - the flow is redirected serially to a set of different hosts.
- file - traffic is captured and sent to a capture file
- ipmap - defines where DHCP or RADIUS traffic is sent for the purpose of mapping subscribers to IP addresses
- group - set of destinations

The general syntax to identify a destination is:

```
destination “destination_name_string” [tee|divert|file|ipmap|group] options
```

The maximum supported number of destinations is 1024 (this includes all types). For example, if you are teeing by IP, you could select up to 1024 different IPs and send each IP to a different destination. Health checking can be used to verify that the hosts in a group of destinations are reachable before sending traffic.

16.3.2.1 Ipmap Destinations

Ipmap destinations define where DHCP or RADIUS traffic is sent for the purpose of mapping subscribers to IP addresses.

Once an ipmap destination has been declared, traffic is sent to it using the tee action.

The syntax is:

```
destination "<name>" ipmap mode ip_rewrite ip <ipaddr> src_ip <ipaddr>
destination "<name>" ipmap mode encapsulation ip <ipaddr> src_ip <ipaddr> port <n>
destination "<name>" ipmap mode ip_encapsulation ip <ipaddr> {src_ip <ipaddr>} {port <n>}
```

Where:

Mandatory

Name	The name of the destination.
mode	Specifies the capture mode configured on the SDE or SPB. For the SDE, it must be ip_encapsulation. For the SPB, it can be either ip_rewrite or encapsulation. This variable must match the capture mode set on the SPB for RADIUS or DHCP. Refer to the *SPB Administration Guide* for more details.
ip	The IP address of the SDE or SPB running the ipusermap application.
port	In encapsulation mode, each DHCP/RADIUS packet is encapsulated inside a UDP header. This option specifies the listening port of the ipusermap application on the SDE or SPB. For the SPB, this variable must match the interface set on the SPB for RADIUS or DHCP . Refer to the *SPB Administration Guide* for more details.

Optional

src_ip	The source IP address which is assigned to the traffic.

16.3.3 Creating a Tee Destination

The first step in configuring teeing is to identify and configure the destination device.

The general syntax is:

```
destination "name" tee options
```

Tee header options are:

```
{next_hop ip}
{ip dst_addr {ip{src_addr ip}{ttl <n>}{tos <n>}{protocol <n>}}
{udp dst_port <n>{src_port <n>}}
{binary "binary_format_string" {expr {expr…}}}}
{payload {packet|ether|public_ip|ip}}
```

When creating a tee destination, keep these points in mind:

- Must begin with {next_hop|ip} followed by any mix of headers or no header, as long as the total length does not exceed 96 bytes (next_hop = 18 bytes, IP = 20/40 bytes, UDP = 8 bytes, binary = variable).
- Tee to capture is limited to a maximum of twelve destinations.
- Teed traffic can be encapsulated by a combination of these headers: IP, UDP, and binary.

The options that can be configured are:

binary Optional header format string in TLC binary format for headers other than IP, or UDP. The size of the fields are specified using these characters:

- C/c - 8-bit integer
- s - 16-bit little-endian integer
- S - 16-bit big-endian integer
- i - 32-bit little-endian integer
- I - 32-bit big-endian integer
- w - 64-bit little -endian integer
- W - 64-bit big-endian integer

expr Arguments for the binary format. The number of expressions is equal to the number of format letters in binary_format_string. Where:

```
expr = {integers|subscriber.id|now|ip_offset|from_subscriber}
```

ip_offset represents the offset in bytes to the start of the ip header from the start of the teed portion of the original packet.

from_subscriber has a value of either 1 or 0, which determines whether the field being teed is coming from or going to the subscriber.

ip dst_addr Destination IP address.

In this IP rewrite example, the payload is required:

```
destination "IpRewrite" tee \
 ip dst_addr 2.0.0.2 \
 src_addr 2.0.0.1 \
 ttl 10 \
 tos 5 \
 protocol 17 \
 payload ip
```

next_hop IP address of the next_hop, used to obtain the Ethernet address. If used, the payload type must be ether.

An example of a MAC rewrite destination is:

```
destination "MacRewrite" tee next_hop 2.0.0.2 payload ether
```

payload Layer of captured packet to include. The payload option must come either before or after all the header options. Specify one of:

- packet - default if next-hop is not used. Entire packet is teed, including the ethernet header.
- ether - default if next-hop is used. Ethernet headers are stripped and the rest of the packet is teed.
- public_ip - encapsulation is stripped down to the public IP headers, which is teed along with the rest of the packet.
- ip - IP header is stripped and the rest of the packet (layers 4 through 7) are teed.

protocol Layer 4 protocol.

Default is 17 (UDP).

src_addr Source address of the IP layer. If omitted, the IP of the outgoing interface of the PTS is used. In the service port case, the PTS 24000/22000/14000 uses control PPU port.

src_port UDP source port number of the packet.

Default is 20000.

tos Marking for the teed packet (IPv4).

Default is 0.

ttl Use to assign a time to live value to the teed packet (IPv4).

Default is 32.

udp dst_port Layer 4 destination port.

Example:
An example of a destination to encapsulate teed traffic is:

```
destination "UdpEncap" tee \
 ip dst_addr 2.0.0.2 \
 udp dst_port 60000 \
 src_port 60001 \
 payload packet
```

Example:
An example of a destination for lawful intercept is:

```
#Declare a destination for each IP address:
destination "TEEGUID1" tee payload packet \
  ip dst_addr 192.168.10.25 \
  udp dst_port 44544  \
  binary "ccSIW" 1 from_subscriber ip_offset subscriber.id \
  now

#Declare 1 healthcheck:
healthcheck "CHECKGUID" ping \
  interval 10seconds \
  timeout 30ms

#Declare 1 destination group to include healthcheck and #destinations above
destination "GROUPGUID" group destinations \
  "TEEGUID1" ... \
  healthchecks "CHECKGUID"
```

Example:
A policy example is:

```
PolicyGroup all {
 if client "intercept"="true" then \
   tee from client destination "GROUPGUID" and \
   tee to client destination "GROUPGUID"
```

```
if server "intercept"="true" then  \
   tee from server destination "GROUPGUID" and \
   tee to server destination "GROUPGUID"
}
```

16.3.4 The Tee Action

Once a tee destination has been defined, the tee action is used to tee packets to the destination(s).

The syntax is:

```
if condition then tee to|from client|server|subscriber|internet destination "<name>"
```

For each flow:

- there can be a maximum of four unique destinations per flow (any combination of file and device to a maximum of four destinations).
- for each destination, there can be a maximum of four node qualifiers, although using two node qualifiers is most common.

The node qualifiers that you can use are:

- from client: all packets are received from the client before any policy has been carried out.
- to server: all packets are transmitted by the PTS in the direction of the server after policy has been applied. This is exactly what the server will receive.
- from server: all the packets are received from the server before policy has been applied.
- to client: all packets are transmitted by the PTS in the direction of the client after policy has been applied. This is exactly what the client will receive.
- from subscriber: all packets are received from the subscriber before any policy has been carried out.
- to subscriber: all packets are transmitted by the PTS in the direction of the internet after policy has been applied. This is exactly what the internet will receive.
- from internet: all packets are received from the internet before policy has been applied.
- to internet: all packets are transmitted by the PTS in the direction of the subscriber after policy has been applied. This is exactly what the subscriber will receive.

See *Node Qualifiers* on page 33.

16.3.4.1 Tee Action Examples

These policy examples demonstrate how to create a tee destination and action.

Example:
This example defines a destination that will create a series of files to capture unknown TCP traffic from the client and all unknown TCP traffic from the server. This destination is then specified in a tee action.

```
destination "destFile1" \
 file "unknownTCPatMyISP" \
 file_size 10000000 \
 file_count 20 \
 overwrite false
if expr (Flow.ApplicationProtocol is null \
 or Flow.ApplicationProtocol = Protocol.unknownTCP) \
 then tee from client destination "destFile1" and \
 tee from server destination "destFile1"
```

Example:

In this example, the destination is specified by IP destination address and UDP destination port with an additional binary header.

```
destination "destUDPEncap1" tee\
   ip dst_addr 1.2.3.4 \
   udp dst_port 50000 \
   src_port 50001 \
   binary "IW" subscriber.id Now

# tee all traffic that is coming from or going to client with network class "tee_needed"
if client class "tee_needed" then \
    tee from client destination "destUDPEncap1" and \
    tee to client destination "destUDPEncap1"
```

Example:
In this example, both regular and teed traffic is evaluated. Traffic is teed only when it falls below 700 Mbps. When the traffic exceeds this amount, no new flows are teed (teeing is not terminated on existing flows).

```
measurement "regular_traffic" bitrate where source port 3 or \
 source port 4
measurement "teed_traffic" bitrate where source port 2 and \
  expr(Flow.IsTeed)
limiter "regular" 700 Mbps measurement "regular_traffic"
limiter "tee" 100 Mbps measurement "teed_traffic"
if client "wanted" = "true"  and \
 expr ((not(Flow.IsReevaluatingPolicy) or \
 (Flow.IsReevaluatingPolicy and Flow.IsTeed)) and \
 not limiter.regular.limit and \
 not limiter.tee.limit) \
 then tee from client destination "destUDPEncap1" \
 and tee to client destination "destUDPEncap1"

measurement "regular_traffic_up" bitrate where source port 3
measurement "regular_traffic_down" bitrate where source port 4

if client "wanted" = "true" and \
    expr((not(Flow.IsReevaluatingPolicy)) or \
    (Flow.IsReevaluatingPolicy and Flow.IsTeed)) and \
    expr(measurement.regular_traffic_up + \
    measurement.regular_traffic_down < 700000000) and \
    expr(measurement.teed_traffic < 100000000) \
    then tee from server destination "destUDPEncap1"
```

16.3.5 Packet Capture to File

Traffic that is captured and sent to a file is typically used for debugging purposes.

A maximum of 12 file destinations for the captured packets can be specified in policy. If additional destinations are specified, the policy reload will fail.

Note:
Sandvine recommends that packet capture to a file should not exceed 10% of the control port's interface rate. If the rate is higher than 10%, the control port can become unresponsive to remote terminal sessions through SSH. You can still manage the element through the serial port.

To configure teeing to a file you must first identify and configure the destination file, then create a tee action that captures specified data to the file.

The syntax is:

```
destination "<name>" file "filename"
 {file_size <n>}
 {file_count <n>}
```

```
{flush_interval <n>}
{overwrite true|false}
```

Where:

name	Name of the destination.
file	File name prefix for the capture file.
file_size	Maximum size of each capture file in bytes. Default is 10MB.
file_count	Maximum number of capture files of the file_size to keep on disk.
overwrite	Indicates whether capture should stop when the maximum number of capture files is reached. If set to true, when the maximum number of capture files have been created, any subsequent captures will cause the oldest capture file to be overwritten. If set to false and the maximum number of capture files have been created, no additional information is captured. Default is true.
flush_interval	Number of seconds to wait until an automatic flush is triggered. The flush_interval ensures that either when the file is full or after a specific period of time, the capture file is closed and renamed so that the contents can be read. Default is 3600.

16.3.5.1 Capture File

The capture filename:

- may include a relative path only. The absolute path will be determined on the receiving end only where the relative path is appended to the default absolute path prefix. Note that current directory (“.”) and parent directory (“..”) are not allowed in the relative path definition.
- the default absolute path is `/d2/var/captures`. The root of the file capture can’t be changed. Note that any empty directories under `/d2/var/captures` are automatically deleted.
- the file name that is specified is used as a prefix and an eight digit number followed by .cap is appended after the prefix is specified. For example, if file unknownUdp is specified, these files will be created:

  ```
  unknownUdp.00000001.cap
  unknownUdp.00000002.cap
  …
  ```

- When files are being written, a temporary file is used. The temporary file can not be read. When a flush occurs, the file is renamed and can be read. For example unknownUdp.00000001.cap.part would be renamed unknownUdp.00000001.cap.

Note:
The PTS default condition stops packet capture to file / over-write existing file whenever there is less than 5G of disk space remaining.

16.3.5.2 Overwrite Option

If overwrite is set to TRUE, when the maximum number of files or disk space is reached, the oldest file is erased. For example, if file_count is set to 10, and overwrite is set to TRUE, when ten files have been written, the oldest file (for example, `unknownUdp.00000001.cap`) is deleted and a new file called `unknownUdp.00000011.cap` is created. When this file is full or a new file is indicated, `unknownUdp.00000002.cap` is deleted and a new file called `unknownUdp.00000012.cap` is created. The ten most recent files are retained.

When overwrite is FALSE, packet capture ceases when the limits are reached. When the maximum number of capture files have been created, it stops writing to disk.

16.4 Divert

16.4.1 Diverting Flows

The divert action causes selected traffic to be diverted (redirected) to a device on the PTS service network instead of being forwarded on the data-intersect network.

You can use divert for targeted advertising, caching (both HTTP and P2P), and URL filtering. The host(s) on the service network are referred to as the divert host(s). These divert hosts typically respond to the traffic, perhaps having modified it. The divert feature is licensable.

The variants of divert are:

- **Half-divert action** - the divert host takes over the role of the original server and the connection with the original server is closed. The PTS re-handshakes a connection with the divert host. The connection to the original server must be re-negotiated by the divert host.
- **Full-divert action** - the divert host comes in-line with the diverted flow with traffic being forwarded from the PTS to the divert host and back to the PTS again before the PTS forwards the traffic to the intended destination. This can be used to send traffic to a device that would be capable of sitting in-line with traffic, such as a transparent proxy.
- **Multi-divert action** - a single flow of traffic is diverted through multiple divert hosts, which apply the required processing to the traffic before it arrives at its destination. Traffic is diverted both to and from the subscriber, in a mirrored sequence of divert hosts.

Note:
Some UDP protocols not diverted completely when diverting by protocol. Policy actions that are based on protocol conditions are not executed until the protocol of the flow has been recognized. If the PTS requires multiple packets of a UDP flow to recognize the protocol, the first packets will not be diverted. This includes Ares UDP and Thunder UDP. The impact depends on how the divert host will react to the flow without receiving the first packet. The PTS continues to inspect the flow as usual, and all subsequent packets are diverted.

16.4.2 Creating a Divert Destination

To divert, you must create a configuration that identifies external equipment as the divert host for the traffic.

You will need to create a divert destination in policy and configure the PTS to receive traffic. See the *PTS Network Configuration Guide* for more information about configuring the PTS for diverting. The maximum supported divert host declarations is 200. The syntax is:

```
destination "<name>" divert
 {ip address}
 {max_connections <n>}
 {mtu mtu_size}
 {reset_server true|false}
 {tcp_syn true|false}
 {interface}
 {replay_safe true|false}
```

Where:

name The name of the destination.

ip The IP address of the destination ARP'd to obtain the Ethernet address to forward packets to the divert host. The IP address must be reachable on the service network segment: diverted traffic can not cross a routed

network. To support this, the IP address must be in the same subnet as one of the IP address on the PTS service network.

Note:
May be specified on either the destination or on each interface, but not both.

max_connections If specified, sets the maximum number of active diverted connections to the destination. New flows above the limit are NOT diverted. If not specified, the number of active diverted flows is unlimited.

This parameter applies to each module. For example, in a PTS 14520, where there are 10 modules, a max_connections of 10 would allow up to 100 connections (10 per module).

mtu The maximum size of packet (counted from the IP layer) that the PTS can transmit to the divert host. The PTS may break the original packet into multiple packets of smaller size if the original diverted packet is larger than mtu_size. If mtu_size is not specified, the PTS uses the MTU of the service port as the default mtu_size.

reset_server Set to true to specify a “half” divert action or set to false for a “full” divert action.

The default is true.

tcp_syn Indicates whether or not the divert destination needs to have the TCP packets in the three-way handshake diverted to it. The parameter is relevant when `reset_server` is set to "false". When `reset_server` is set to "true", PTS establishes three-way handshake with the divert host.

The default is false.

interface Defined as:

```
interface interface_name
    {ip address}
    {vlan number}
    {vlan_label label}
```

And where:

- **vlan** is the VLAN number over which to send diverted traffic. Packets diverted to this destination are tagged with the configured VLAN. The VLAN number must be between 150 and 3500 inclusive. The maximum number of VLANs that can be configured is 400. vlan and vlan_label options may not be used simultaneously. May be specified on either the destination or on each interface, but not both.
- **vlan_label** is similar to the vlan option, the label specifies the VLAN number over which to send diverted traffic. The label must be configured via the SetVlanLabel configuration function. Using the vlan_label option is advantageous when different elements in a cluster executing the same policy wish to use different VLANs for diverted traffic. The maximum number of VLANs that can be configured is 400. vlan and vlan_label options may not be used simultaneously. May be specified on either the destination or on each interface, but not both.

replay_safe This optional parameter indicates that contractually the destination will not modify the initial packets of a flow. If this option is "false", flows identified late are not diverted to the destination.

The default is "false".

A divert destination can be sent to one or two interfaces of the divert host. To send traffic for both directions of a flow to the one interface of the divert host:

```
destination "<name>" divert ip <ip-address>
    {max_connections <number>}
    {reset_server true|false}
    {tcp_syn true|false}
```

Or, to explicitly send traffic to each interface of the divert host, explicitly name each one:

```
destination "<name>" divert reset_server false
    {max_connections <number>}
    {tcp_syn true|false}
```

```
interface "<ifname>" ip <ip-address>
   interface "<ifname>" ip <ip-address>
```

Note that when a divert destination has two interfaces, the `reset_server` variable must be set to 'false'.

16.4.2.1 Creating a Divert Sequence Destination

Divert sequence destinations are similar to divert destinations except that they define a sequence of divert destinations through which diverted traffic passes serially.

Traffic is diverted to a divert_sequence destination using the divert action. See *Creating a Divert Destination* on page 176 for details on divert_options and interface_options.

The maximum supported divert sequences is 200.

The syntax is:

```
destination name divert_sequence destinations child1 {child2 ...}
    {divert_options}
    {interface interface_name {interface_options}}
```

Where:

name	The name of the destination.
divert_sequence	Redirects the flow in a sequence to a set of different hosts..
child	The name of a previously declared divert destination. Providing a properly configured list of child destinations can help ensure consistency and correctness of a divert policy. You must specify at least one child destination.
interface	Identifies the name of the destination interface.
divert_options and interface_options	The same options that can be specified on divert destinations can be specified on divert_sequence destinations with these exceptions: • The ip clause is not allowed on interface option of divert sequence destinations. • The divert sequence's child destinations cannot have interface option specified. • The vlan or vlan_label option is mandatory for divert sequence destinations. • If interfaces are specified, then the vlan or vlan_label options are mandatory for each destination interface.

16.4.3 The Divert Action

Once a divert destination has been defined, the divert action is used to divert packets to the destination(s).

The syntax is:

```
if <condition> then divert destination "<divert_name>"|"<group_dest_name>"
```

If a specific destination is named, traffic will be forwarded to the destination, provided that it has responded to the ARP messages. If a group is named, traffic will be load shared to one of the hosts in the group that is passing the healthchecks. Load sharing is done on the basis of subscriber IP address and is consistent across elements in a cluster. A divert destination must reside on the local subnet of (same layer2 network as) the PTS service port(s).

See *Destination Groups* on page 165 for information on how to create destination groups.

16.4.3.1 Full-divert Example

This example is a policy to divert HTTP and peer-to-peer traffic to a third party caching solution.

In this example the third party caching solution is PeerApp (Sandvine supports other vendors as well). This policy diverts specific protocols and maintains a health check on the caching devices. The PTS will reinsert the traffic once it has been processed by the caching device.

```
# Each specific PeerApp box needs to be a distinct "destination"
destination "peerapp1" divert ip 192.168.22.2 \
   reset_server false mtu 1500 tcp_syn true
destination "peerapp3" divert ip 192.168.22.4 \
   reset_server false mtu 1500 tcp_syn true
# This defines the health checking type and parameters.
healthcheck "check1" ping interval 2 seconds timeout 500ms
# This defines a "group" of destinations for load-balancing purposes.
# It also enables the health checking.
destination "peerapp" group destinations "peerapp1" "peerapp3" \
   healthchecks "check1"
# This is the rule to select which traffic goes to an element of the
# group subject to load-balancing.

# If Flow is from subscriber and layer4protocol TCP then divert destination "peerapp"
if expr(Flow.Client.PortRole = PortRole.Subscriber) and \
(protocol "http_get" or protocol "http" or protocol "edonkey" or \
 protocol "gnutella" or protocol "bittorrent" or \
 protocol "youtube" or protocol "metacafe" or \
 protocol "googlevideo") then divert destination "peerapp"
```

This example illustrates a policy that can be used to split diverted traffic between two different divert destinations.

```
# Create a divert destination that maps to two interfaces
destination "Divert0" divert reset_server false \
 tcp_syn false interface "A" ip 192.168.1.1 interface "B" ip 192.168.1.2
# Create a destination group to contain the above destination
destination "Group0" group destinations "Divert0"
# Divert http traffic to two separate interfaces
# Client traffic is sent to interface A, while server traffic is sent to interface B
if expr(Flow.Client.PortRole = PortRole.Subscriber) and protocol "http" then \
 divert destination "Group0" \
 from client interface "A" from server interface "B"
```

This example is a policy to divert based on the content type of a server:

Note:
`Flow.Stream.SaveData` is set to 'true' to allow the PTS to save the application layer content prior to divert.

```
##define the full divert destination
destination "fullDivert" divert \
                ip 2.0.0.1 \
                reset_server false \
                mtu 1500 \
                tcp_syn true \
                replay_safe true

healthcheck "check1" ping interval 2 seconds timeout 500ms

destination "fullDivertGroup" group destinations "fullDivert" healthchecks "check1"

PolicyGroup expr(Flow.SessionProtocol=Protocol.HTTP and \
                 Flow.Server.Stream.Http.TransactionNumber=1)
{
                #divert for flashvideo or text page
                if expr(Flow.ApplicationProtocol = Protocol.FlashVideo or \
                          Flow.Server.Stream.Http.ContentType = "text/html") \
                then \
                                divert destination "fullDivertGroup"

               # save data when application protocoltype is unknown or content type is unknown

                if expr(Flow.ApplicationProtocol is null or \
                        Flow.Server.Stream.Http.ContentType is null) \
```

```
                then \
                                analyze analyzer "http" and \
                                set Flow.Stream.ReadOnly = false and \
                                set Flow.Stream.SaveData = true
}
```

Note:
A flow starts as read-only in its first policy run, but if at any point the `Flow.Stream.ReadOnly` is not set to 'false', the flow will become read-only forever. When the flow is not read-only, the PTS buffers packets to reassemble the stream to analyze HTTP. If a flow continues in non read-only mode, latency is added to the HTTP connection. This is a problem in case of lost or reordered packets. Set the `Flow.Stream.ReadOnly` value to "true" in the SandScript as soon as possible.

16.4.3.2 Multi-divert Example

This simple policy example illustrates a typical setup for diverting HTTP flows.

It configures divert destinations that use VLANs 2500 and 2600. In this example a single destination sequence is defined containing a set of three divert hosts. The sequence is wrapped in a destination group and has all three supported types of health checks applied to it. The HTTP traffic is diverted on the correct VLAN depending on which side of flow is the client and server side. For more information about multi-divert see the *PTS Network Configuration Guide*.

```
# Define the divert destinations for our sequence
#
destination "DivertHost1" divert ip 1.0.0.0 tcp_syn false reset_server false
destination "DivertHost2" divert ip 1.0.0.1 tcp_syn false reset_server false
destination "DivertHost3" divert ip 1.0.0.2 tcp_syn false reset_server false

# Define the sequence of divert hosts using the VLAN numbers
#
destination "DivertSeq" divert_sequence tcp_syn false reset_server false \
    destinations "DivertHost1" "DivertHost2" "DivertHost3"\
    interface "fromClient" vlan 2500 \
    interface "fromServer" vlan 2620

# Define the health checks to be used for the divert sequence and its
# destinations. Health checks are optional but highly recommended.
# Ping and HTTP health checks operate on the ip(s) specified for each divert
# destination. Inline health checks operate on the vlans specified for each
# divert sequence.
#
healthcheck "Ping" ping interval 5seconds timeout 1second retry 2 retry_failure 2
healthcheck "Inline" inline interval 5seconds timeout 1seconds retry 2 retry_failure 2
healthcheck "Http" http interval 5seconds timeout 1second retry 2 retry_failure 2 path "/status"

# Define a destination group with a single divert sequence using both
# inline and ping health checks.
#
destination "Group" group destinations "DivertSeq" healthchecks "Ping" "Inline" "Http"
# Define the conditions to be used for determining what flows to divert.
#
if expr(Flow.SessionProtocol = Protocol.Http) then \
    divert destination "Group" \
    from client interface "fromClient" \
    from server interface "fromServer"
```

17

Diameter Stack Configuration

17.1 The Diameter Stack

The Diameter stack is enabled by default when Diameter peers are correctly configured.

The information in this guide is for advanced configuration, debugging and informational purposes, and is therefore not required information for a typical deployment.

The Diameter peer configuration file is /usr/local/sandvine/etc/diam_peer_config.xml. Policy exists to handle Diameter messaging; see *Diameter* on page 46.

17.2 Diameter Stack Base Protocol Messages

A session begins when the Diameter stack sends a Capabilities Exchange Request (CER) to one of its configured Diameter peer servers.

If an acceptable Capabilities Exchange Answer (CEA) is received, the stack moves into the connected state and begins to send periodic Device Watchdog Request (DWR) and expects to receive Device Watchdog Answer (DWA) Messages in response. Any DWR's received by the Diameter stack are responded to by an appropriate DWA. Disconnect Peer Requests (DPR) are currently handled by closing the connection, no Disconnect Peer Answer (DPA) is sent. The Diameter stack does not currently generate DPR requests and thus does not handle DPA messages.

For outgoing message definitions, mandatory AVPs are included with every outgoing message. Optional AVPs may or may not be included with the outgoing message, depending on the stack configuration.

For incoming message definitions, mandatory AVPs are required with every incoming message. Optional-C AVPs are used by the dictionary, but incoming messages can include any number of instances (including zero) in the message. Optional AVPs are listed in the RFC, but the diameter stack does not explicitly check or use them.

Further information is available in:

- RFC 3588 - Section 2.5 Connections vs. Sessions
- RFC 3588 - Section 3.2 Command Code ABNF specification

17.2.1 CER Messages

A Capabilities-Exchange-Request (CER) is sent by a diameter peer to initiate a connection to another diameter peer.

Outgoing CER Messages

The Diameter stack initiates a session by sending a CER message with this format:

AVP	Category	Details
Origin-Host	Mandatory	Configurable via changing LocalIdentify
Origin-Realm	Mandatory	Configurable via changing LocalRealm
Host-IP-Address	Mandatory	Configurable via changing LocalIp
Vendor-Id	Mandatory	Configurable via VendorId
Product-Name	Mandatory	Configurable via ProductName
Origin-State-Id	Mandatory	Not configurable
Inband-Security-Id	Mandatory	Not configurable

AVP	Category	Details
Firmware-Revision	Mandatory	Not configurable
Supported-Vendor-Id	Optional	Configurable via dictionary definition
Auth-Application-Id	Optional	Configurable via dictionary definition
Acct-Application-Id	Optional	Configurable via dictionary definition
Vendor-Specific- Application-Id	Optional	Configurable via dictionary definition

Incoming CER messages

The Diameter stack drops any incoming CER messages.

17.2.2 CEA Messages

A Capabilities-Exchange-Answer (CEA) is sent by a Diameter peer in response to a CER message.

Outgoing CEA Messages

Since the Diameter stacks drops any incoming CER messages, no CEA messages are sent.

Incoming CEA messages

The AVPs listed in the table are mandatory. If any are missing in an incoming CEA, the Diameter stack will close the connection. If there is no set of common supported Application Ids, the connection is also closed. The Diameter stack expects any incoming CEA messages to be of the format described in the table.

AVP	Category	Details
Result-Code	Mandatory	Must be DIAMETER_SUCCESS
Origin-Host	Mandatory	
Origin-Realm	Mandatory	
Vendor-Specific-Application-ID	Optional-C	Used to determine negotiated applications.
Auth-Application-Id	Optional-C	Used to determine negotiated applications.
Acct-Application-Id	Optional-C	Used to determine negotiated applications.
Host-IP-Address	Optional	
Vendor-IP	Optional	
Product-Name	Optional	

17.2.3 DWR Messages

A Diameter peer may send a Device-Watchdog-Request (DWR) if no messages have been exchanged for some time.

Outgoing DWR Messages

The Diameter stack periodically sends DWR messages with this format:

AVP	Category	Details
Origin-Host	Mandatory	Configurable via changing LocalIdentify.
Origin-Realm	Mandatory	Configurable via changing LocalRealm.
Origin-State-Id	Mandatory	Not configurable.

Incoming DWR Messages

The incoming DWR message format is:

AVP	Category	Details
Origin-Host	Optional	
Origin-Realm	Optional	
Origin-State-Id	Optional	

17.2.4 DWA Messages

A peer that receives a DWR message must respond with a Device-Watchdog-Answer (DWA).

Outgoing DWA Messages

The Diameter stack responds to an incoming DWR message with a DWA message with this format:

AVP	Category	Details
Origin-Host	Mandatory	Configurable via changing LocalIdentify
Origin-Realm	Mandatory	Configurable via changing LocalRealm
Origin-State-Id	Mandatory	Not configurable
Result-Code	Mandatory	Not configurable. Always set to DIAMETER_SUCCESS.

Incoming DWA Messages

Upon reception of a DWA, the Diameter stack updates its connection timeout. The format of the incoming DWA must only have a valid header and command code. Thus, the Diameter stack expects any incoming DWA message to be of this format:

AVP	Category	Details
Result-Code	O	
Origin-Host	O	
Origin-Realm	O	
Error-Messages	O	
Failed-AVP	O	
Original-State-Id	O	

17.2.5 Outgoing DPR Messages

Upon reception of a Disconnect-Peer-Request (DPR) message, the connection is immediately closed, no outgoing DPA message is sent in response. The format of the incoming DPR must only have a valid header and command code. Thus, the Diameter stack expects any incoming DPR message to be of the format:

AVP	Category	Details
Origin-Host	O	
Origin-Realm	O	
Disconnect-Cause	O	

18

Using the Network Protection Module

18.1 Network Protection Overview

The network protection module mitigates deleterious traffic.

The terms used to describe such traffic are:

- event is a group of packets thought to be malicious. It is an indication of something notable.
- detection is a group of events over time.
- attack is a detection that is considered malicious.

The modules that implement network protection are:

- PTS daemon (PTSD) monitors the network traffic to find malicious traffic based on network demographics. Information obtained can be passed to the CND for automatic mitigation and logging to the network protection database.
- Central Node Daemon (CND) aggregates cluster-wide detection events and distributes mitigation actions.
- Border Gateway Protocol (BGP) daemon (optional) implements network class subnet retrieval and handles the selective in-line feature.

18.1.1 Malicious Traffic Overview

The types of malicious traffic that can be detected and potentially mitigated are:

- Denial of service
- Malware
- DNS Attacks or Defense
- Spam

Denial of Service:

Traffic Type	Description	Detected Host
User bandwidth attack	User bandwidth attacks attempt to disable a host by overloading it with data. When the target is under attack, usually the inbound download pipe is saturated and no legitimate traffic can be accommodated. Bandwidth attacks are usually distributed.	Target of attack. dst-ip
SYN flood attack	A SYN flood attack is identified as a large number of incomplete TCP connection attempts to a single port on a single host.	Target of attack. dst-ip dst-port

Malware:

Traffic Type	Description	Detected Host
Address scans	Address scans are defined as a single host initiating TCP, UDP, or ICMP flows to many destination hosts on a specific port. Address scans are strong indicators of worm activity. If an external host is address scanning subscriber addresses, only address scans that exceed n times the threshold, where n is the number of modules in the PTS (for example, 10 for PTS 14520), will be detected.	Initiator of attack. src-ip dst-port
Flow flood attacks	Flow flood attacks are defined as a single host attempting to connect from multiple ports to another host on a single port.	Initiator of attack. src-ip

Traffic Type	Description	Detected Host
		dst-ip
Malicious traffic matched by signature	Signature match refers to worms and trojans which typically can be detected by matching specific sequences of bytes in the traffic.	Can be either the target of the attack or the initiator of the attack depending upon the signature. src-ip, dst-ip or both src-port, dst-port or both (any combination of ip and port)
User bandwidth transmit attack	User transmit bandwidth attacks typically occur when a subscriber has modified their modem to transmit more bandwidth than it would otherwise be permitted to transmit. This can affect other subscribers that are sharing resources with this malicious subscriber.	Initiator of attack. src-ip

DNS attacks or defense

Traffic Type	Description	Detected Host
Domain flood	A domain flood occurs when a host attempts to query a given domain such that the query rate exceeds a given threshold.	Initiator of attack. src-ip
DNS request flood	A request flood occurs when a host issues queries of a particular request in an attempt o disrupt the function of the DNS server.	Initiator of attack. src-ip
DNS outstanding sessions	To protect the function of DNS servers, the PTS offers the ability to limit the number of outstanding sessions each host can have active at any given time. This prevents a host from overwhelming a DNS server with more requests than it can handle.	Initiator of attack. src-ip

Spam

Traffic Type	Description	Detected Host
Outgoing spam	Spam transmissions are typically initiated by hosts infected by trojans which are sending unsolicited emails.	Initiator of attack. src-ip

18.1.2 Mitigation in a Clustered Deployment

The CND (Central Node Daemon) can centrally mitigate traffic across a cluster of elements.

One CND must be elected as master to perform this task. A master election takes 60 seconds, by default. The PTS may be detecting traffic, but attack notifications are not made until the master is elected. Therefore, attack mitigation may seem delayed.

18.2 Modifying Network Protection Policy

To modify network protection policy:

1. From any element in the cluster, open the `/usr/local/sandvine/etc/policy.cluster.conf` file in a text editor.
2. Add or modify network protection rules as required.

   Each rule must have a unique rule number. This rule number allows the software to reload only rules that have been added or modified.
3. To load the new policy, run: `reload`.
4. To verify the rules are correct, run: `show policy attack rules`

   For example:

```
PTS> show policy attack rules
Id  Active  Start End Rule
--- ------- ----- --- ------------------------------------------------
0.1 [true]            syn-flood summary-period 5m
1.1 [true]            address-scan threshold 5 period 5 enable on
1.3 [true]            address-scan sample-period 1m timed-host-percent 4..
2.2 [true]            address-scan limit-host-packets
2.4 [false]           address-scan bad-param limit-host-packets
```

5. If the Active flag is false for any of the rules, run: `show policy attack rules status`.

   This CLI command displays each rule along with a status column. Any errors detected when parsing are displayed under the status column as well as being recorded in the `/var/log/svlog` file. A blank status column indicates that the rule was parsed and implemented correctly. See *Verifying Rules Status* on page 192
6. Verify that the new rule(s) did not have any negative effects on other elements in the cluster.
7. If the new rule(s) is functioning correctly, run: `/usr/local/sandvine/bin/wdtm --propagate`

18.2.1 File Precedence

Rules in policy.cluster.conf always take precedence over rules in policy.sandvine.conf as policy.cluster.conf is more specific in nature.

Rules in policy.local.conf always take precedence over policy.cluster.conf. This is because the local policy file is more specific than the cluster policy file. The rules contained in the local policy file affect just the element on which the file resides.

18.2.2 Verifying Rules Status

Use the `show policy attack rules status` CLI command to view which rules are active and the status of the rules.

Note that it is possible for an error message to be repeated multiple times for a single rule. This indicates that processes are attempting to implement the rule but have detected an error. An example of the output of the status command is:

```
PTS> show policy attack rules status
Id  Active  Status
--- ------- -------------------------
0.1 [true]
1.1 [true]
1.3 [true]
2.2 [false] Unknown option `bad-param`
2.4 [true]
```

Where:

- Id displays two numbers. The first digit indicates where the rule came from, and the second digit indicates the rule number. The first digit can be one of:
  - 0: Sandvine policy file (policy.sandvine.conf)
  - 1: cluster policy file (policy.cluster.conf)
  - 2: local policy file (policy.local.conf)
  - 3: rules implemented at runtime
- Active indicates whether the rule is in effect and has no errors (true) or whether the rule was not active when the status command was run (false). A rule is not active if there are errors or if the rule is time dependant and as a result is not currently active.
- Status displays any error message associated with the rule.

If required you can also drill down on a specific rule by indicating the rule identifier. For example:

```
PTS> show policy attack rules 2.4
Attack Rule: 2.4
----------------
Active            : [false]
Start             :
End               :
Order             : 1
Status            : Unknown option `bad-param`
ActiveActions     : 0
TotalActions      : 0
Packets           : 0
Bytes             : 0
DroppedPackets    : 0
DroppedBytes      : 0
DroppedActions    : 0
Rule              : address-scan bad-param limit-host-packets
```

You can also verify the rule by running: `show policy attack rules counts`. For example:

```
PTS> show policy attack rules counts
Id  ActiveActions TotalActions  Packets  Bytes DroppedPackets DroppedBytes DroppedActions
--- ------------- ------------  -------- ----- -------------- ------------ --------------
304             0            0         0           0              0            0
 0
310             0            0         0           0              0            0
 0
350             0         2251     614703   41265306            216        25172
 0
351             0          261  27399186  1845455840          47974      3562455
 0
```

18.3 Network Protection Rules

Rules for network protection are supplied and updated by Sandvine via the `policy.sandvine.conf` file.

If additional rules are required, they can be added to the `policy.cluster.conf` file on an element. The Sandvine provide rules are:

- *detection-config* rules configure algorithms which are used to find events
- *aggregator-config* rules determine how events are grouped and filtered to determine if an attack is occurring
- *wmd-rules* are used to configure mitigation to block malicious traffic
- monitor-config rules are used to configure non-detection related features such as QoE and VoIP

The syntax for network protection rules are:

```
aggregator-config <rule-number> {priority}{configuration <values>}
detection-config <rule-number> {priority} {configuration <values>}
monitor-config <rule-number> {priority} {configuration <values>}
wmd-rule <rule-number> {priority} {start <date-time>}{end <date-time>}{condition}{action}
```

Where:

end <date-time> Indicates the date and time when the rule will be automatically terminated. If start and end are not specified, the rule is always active. Format: yyyy-mm-dd hh:mm:ss Example: end {2007-04-10 23:59:59}

priority <n> A number from 1 to 10 with 1 being the highest priority. The rule priority does not impact the order in which rules are applied. The priority determines which rules are implemented. A rule with a priority of 1 is extremely important and must be implemented even if the performance penalty is significant. A rule with priority 10 can likely be ignored if necessary.

You can choose to implement rules above a specific priority. If no priority is specified, a default priority of 1 is assumed. This ensures that all rules created by the customer are enabled. Other priority levels are reserved for Sandvine standard mitigation policies and should not be used. Priority 1 rules take precedence over any policies set by the Sandvine Security Operations team.

The default priority is set using the `set config policy network-protection minimum-rule-priority` CLI command.

rule-number A rule number unique within a file. The rule number is a label. It is used to determine if/when the content of rules have changed and improves the ability of the network protection feature to reload rules without affecting traffic. This number is used to match detection to mitigation and correlation.

start date-time Indicates when the rule is started automatically. If start and end are not specified, the rule is always active. Format: yyyy-mm-dd hh:mm:ss Example: start {2007-04-10 00:00:00}

18.4 Detection Configuration (detection-config)

The detect-config rule detects different types of malicious traffic.

18.4.1 Excluding Hosts from being Detected

An angel-ip address is used to specify a host to exclude from all detections.

This can be used to prevent specific ISP controlled servers and so forth from being falsely detected. For example, excluding email servers prevent them from showing up on spam reports. Note that this also prevents them from being protected from attack.

The syntax is:

```
detection-config <rule-number> {angel-ip address <ip>}
```

Where:

angel-ip address Specifies address which is excluded from detections.

This example excludes the IP address 10.3.10.2 from all detections.

```
 detection-config 1 {angel-ip address 10.3.10.2}
```

18.4.2 User Bandwidth

The user-bandwidth rule attempts to find internal subscribers (in finite network classes) who are being attacked by high-bandwidth traffic.

The user-bandwidth rule does not detect internal subscribers who are attacking nor does it find external hosts that are being attacked.

To identify subscribers you can use either a network class or an IP address. If a network class is specified, then the enable parameter is supported and the syntax is:

```
detection-config <rule-number> {user-bandwidth
 {class all-internal|<string>}
 {event-forward-period <n>}
 {period <n>} {threshold <n>} {enable on|off}}
```

If an IP address is specified, then the enable parameter is NOT supported because it is always implicitly enabled and the syntax is:

```
detection-config <rule-number> {user-bandwidth
 {address ip|all}
 {event-forward-period <n>}
 {period <n>} {threshold <n>}}
```

The parameters are:

class Specifies which network class is examined for the analysis. Options are:

- all-internal: internal network classes only.
- string: a single network class or policy class from the subnets.txt file.

Default is all-internal.

address Specifies which addresses are examined for analysis. Options are:

- ip: the specified IP address is examined.
- all: all IP addresses are examined (perhaps limited by the network class selection).

Default is all

enable Set to on to enable the rule, off to disable it. Note that this is not supported if an individual IP address is specified.

period Time over which the analysis is performed. This starts whenever the network protection policy is read. Specified as: `s|seconds m|minutes h|hours d|days`. For example: `10s 30seconds 6h 12hours`

Allowable range is 1-30, default is 30seconds.

event-forward-period Determines how often events are passed between PTSD and CND. Value must be greater than the period and never 0. Sandvine recommends setting this to a minimum of 10 seconds.

threshold If specified, a report is generated only if the combined bitrate of all packets to a user exceeds the specified numbers (bps) averaged over the period.

Valid values are in the range 8192 bps - 1000000000 bps. A value with no specified rate is assumed to be bps. Units that can be specified include: `{k|K|M}` (or nothing) followed by `bps|bit/s|bits/s|b/s|byte/s|bytes/s|B/s`

This policy example classifies any IP address within the network class named `dsl`, that receives a sustained rate of 10 megabits over a 30 second period, as the target of a user-bandwidth attack.

```
detection-config 1 {user-bandwidth class dsl period 30 threshold 10000000 enable on}
```

18.4.3 Transmitted Bandwidth

The user-tx-bandwidth rule detects bandwidth from a source for each subscriber in the specified network classes.

If a src-class is specified, the enable parameters is supported and the syntax is:

```
detection-config <rule-number> {user-tx-bandwidth
 {class all-internal|<string>}
 {event-forward-period <n>} {period <n>}
 {threshold <n>} {enable on|off}}
```

If an IP address is specified, the enable parameter is NOT supported because it is always implicitly enabled and the syntax is:

```
detection-config <rule-number> {user-tx-bandwidth
 {address ip|all}
 {event-forward-period <n>} {period <n>}
 {threshold <n>}}
```

The parameters are:

class Specifies which network class is examined for the analysis. Options are:

- all-internal: internal network classes only.
- string: a single network class or policy class from the subnets.txt file.

Default is all-internal.

address Specifies which addresses are examined for analysis. Options are:

- ip: the specified IP address is examined.
- all: all IP addresses are examined (perhaps limited by the network class selection).

Default is all

enable Set to on to enable the rule, off to disable it. Note that this is not supported if an individual IP address is specified.

period Time over which the analysis is performed. This starts whenever the network protection policy is read. Specified as: `s|seconds m|minutes h|hours d|days`. For example: `10s 30seconds 6h 12hours`

Allowable range is 1-30, default is 30seconds.

event-forward-period Determines how often events are passed between PTSD and CND. Value must be greater than the period and never 0. Sandvine recommends setting this to a minimum of 10 seconds.

threshold If specified, a report is generated only if the combined bitrate of all packets to a user exceeds the specified numbers (bps) averaged over the period.

Valid values are in the range 8192 bps - 1000000000 bps. A value with no specified rate is assumed to be bps. Units that can be specified include: `{k|K|M}` (or nothing) followed by `bps|bit/s|bits/s|b/s|byte/s|bytes/s|B/s`

For example:

```
detection-config 3 {user-tx-bandwidth src-class all-internal \
 period 1 threshold 4000000 enable on \
 event-forward-period 10}
```

18.4.4 SYN Flood

SYN floods are another method malicious programs use to disrupt the operation of a victim's computer service. The syn-flood action detects IP addresses (users) that are the target of such an attack (contrast with flow floods, which will find the attacker).

SYN floods are a TCP attack. A SYN flood is detected when large numbers of TCP connection attempts are made to a single IP address/port combination and the majority of these connection attempts fail. The failed connections may be due to the host rejecting the connections or simply being overwhelmed by the volume. Thus, each period we calculate the ratio of incomplete connections to total connection attempts. A high ratio of incomplete connections is a sign that the host either does not wish to connect or that it has been overwhelmed. The ratio, however, is only meaningful when a certain number of connections have been attempted. Thus, the samples parameter indicates the number of connection attempts that must be made before the ratio is considered valid.

For example: one failed connection attempt in an interval is 100% failure rate - which will certainly meet the threshold criteria - but a single connection attempt could hardly be considered as an attack. Thus, the samples parameter should be set to a reasonable value (such as 15 connection attempts per second), so that the ratio becomes meaningful.

The rule syntax is:

```
detection-config <rule-number> {syn-flood
 {class all|all-internal|all-external|<string>}
 {port <n>|all} {period <n>}{threshold <n>}{samples <n>}
 {enable on|off}
```

The parameters are:

class Specifies the network class that is examined to determine if it is under attack. Options are:

- all: all network classes. The default.
- all-internal: internal network classes only.
- all-external: external network classes only.
- string: a single network class or policy class name from the subnets.txt file.

Default is all.

port Specifies which IP ports are examined for analysis. Only packets from the specified port (TCP/UDP) or type (ICMP) will be considered. Specify a port number or `all` for all ports.

enable Set to on to enable the rule, off to disable it. Note that this is not supported if an individual IP address is specified.

period Time over which the analysis is performed. This starts whenever the network protection policy is read. Specified as: `s|seconds m|minutes h|hours d|days`. For example: `10s 30seconds 6h 12hours`

Allowable range is 1-30, default is 30seconds.

sample The minimum number of connections plus connection attempts that must occur per second for a meaningful connection/failed connection ratio to be calculated. See Notes section for additional details.

Recommended value is at least 15 (for both internal and external hosts).

threshold If specified, a report is generated at the end of each period, if the number of connections to a single destination IP per second on the specified destination port is greater than this threshold.

Valid values are in the range 1 - 10000000. Sandvine recommends setting this value to 15 for internal hosts and 10 for external hosts.

In this example, any local host that meets the criteria is considered to be the target of a SYN flood. The criteria is at least 200 (20/sec * 10 sec) connection attempts to any port in 10 seconds, 90% of which fail.

```
detection-config 1 {syn-flood class all-internal port all \
 period 10 threshold 90 samples 20 enable on}
```

This second example modifies the threshold down to 50% for port 80, which is presumably a commonly attacked port.

```
detection-config 1 {syn-flood class all-internal port 80  \
 period 10 threshold 50 samples 20 enable on}
```

18.4.5 Address Scan

The address-scan action is used to find hosts which are actively performing address scans. Hosts identified by this detection often have been infected by a worm or virus.

The syntax is:

```
detection-config <rule-number> {address-scan
 {class all|all-internal|all-external|<string>}
 {transport all|tcp|udp|icmp} {port <n>|all}
 {period <n>}{threshold <n>}
 {enable on|off}}
```

The parameters are:

class Specifies the network class that is examined to determine if it is under attack. Options are:

- all: all network classes. The default.
- all-internal: internal network classes only.
- all-external: external network classes only.
- string: a single network class or policy class name from the subnets.txt file.

Default is all.

port Specifies which IP ports are examined for analysis. Only packets from the specified port (TCP/UDP) or type (ICMP) will be considered. Specify a port number or `all` for all ports.

sample The minimum number of connections plus connection attempts that must occur per second for a meaningful connection/failed connection ratio to be calculated. See Notes section for additional details.

Recommended value is at least 15 (for both internal and external hosts).

transport Specifies the transport protocol. Can be one of: tcp|udp|icmp|all. Default is all.

enable Set to on to enable the rule, off to disable it. Note that this is not supported if an individual IP address is specified.

period Time over which the analysis is performed. This starts whenever the network protection policy is read. Specified as: `s|seconds m|minutes h|hours d|days`. For example: `10s 30seconds 6h 12hours`

Allowable range is 1-30, default is 30seconds.

threshold Specifies the "reporting threshold" for the analysis. A report is generated (at the end of the period) only if the value (detection specific units) is greater or equal to the threshold (detection specific units). Is the number of

distinct IP addresses contacted by a single host using the same destination port per second that should be considered an address scan.

Sandvine recommends the value for internal hosts is 15 addresses connected per second from the same host. Recommended value for external hosts is 10 addresses connected per second from the same host.

For this example, any local host that connects to 15 other hosts per second on the same port for 5 seconds is considered malicious.

```
detection-config 1 {address-scan class all-internal \
 transport tcp port all period 5 threshold 15 \
 enable on}
```

This example adjusts the threshold value for port 80 (HTTP) up somewhat to avoid false positives for this commonly used port.

```
detection-config 2 {address-scan class all-internal \
 transport tcp port 80 threshold 30 enable on}
```

Consider this policy:

```
detection-config 1 {address-scan class all-internal period 30 \
 threshold 15 enable on}
wmd-rule 2 {address-scan class all-internal} {limit-packets \
 src-ip ?src-ip dst-port ?dst-port}
```

A host detected and mitigated by this policy by sending an TCP address-scan (for example) is reported as a malicious host with a transport protocol of TCP. As well, the mitigated host count in the reports specify a transport of TCP. However, because the transport is not specified in the mitigation rule, the mitigated bytes/packets reported fall under a transport of unknown. This can cause confusion in the reports because it becomes possible for a report to show mitigated bytes/packets for transport unknown, but no mitigated hosts. Finding the mitigated host counts would require drilling down further into more specific transports.

This can be remedied by modifying the policy:

```
detection-config 1 {address-scan class all-internal period 30 \
 threshold 15 enable on}
 wmd-rule 2 {address-scan class all-internal} {limit-packets \
 src-ip ?src-ip dst-port ?dst-port transport ?transport}
```

18.4.6 Flow Flood

The flow-flood detection is used to find hosts which are actively performing flow-floods. Hosts identified by this detection may be infected by a worm or virus.

A common distributed denial of service attack is to have an infected host bombard a specific target address (victim) with many connections. This bombardment (from possibly thousands of infected hosts) causes the resources of the victim address to be exhausted very quickly, thus denying legitimate use.

The syntax is:

```
detection-config <rule-number> {flow-flood
 {class all|all-internal|all-external|<string>}
 {transport all|tcp|udp|icmp} {port <n>|all} {period <n>}
 {threshold <n>}
 {enable on|off}}
```

The parameters are:

class Specifies the network class that is examined to determine if it is under attack. Options are:

- all: all network classes. The default.
- all-internal: internal network classes only.
- all-external: external network classes only.
- string: a single network class or policy class name from the subnets.txt file.

Default is all.

port Specifies which IP ports are examined for analysis. Only packets from the specified port (TCP/UDP) or type (ICMP) will be considered. Specify a port number or `all` for all ports.

transport Specifies the transport protocol. Can be one of: tcp|udp|icmp|all. Default is all.

enable Set to on to enable the rule, off to disable it. Note that this is not supported if an individual IP address is specified.

period Time over which the analysis is performed. This starts whenever the network protection policy is read. Specified as: `s|seconds m|minutes h|hours d|days`. For example: `10s 30seconds 6h 12hours`

Allowable range is 1-30, default is 30seconds.

threshold If specified, a report is generated at the end of each period, if the number of connections to a single destination IP per second on the specified destination port is greater than this threshold.

Valid values are in the range 1 - 10000000. Sandvine recommends setting this value to 15 for internal hosts and 10 for external hosts.

For this rule, any local host that connects 15 times per second to a single host on the same server port over 5 seconds is considered malicious.

```
detection-config 1 {flow-flood class all-internal \
 transport tcp port all period 5 threshold 15 enable on}
```

This rule adjusts the threshold value for port 80 (HTTP) up somewhat to avoid false positives for this commonly used port.

```
detection-config 2 {flow-flood class all-internal \
 transport tcp port 80 threshold 30 enable on}
```

18.4.7 Static Signature

The static signature detection re-configures network protection to load the pattern for a signature specified by pattern.

This is used to detect the signature, create reports in the database and in the case of a worm signature, send events to the aggregator that will mitigate a host.

The syntax is:

```
detection-config <rule-number> {signature-match
 {pattern <string>|all} {enable on|off}}
```

The available signatures can be identified using the `show policy attack rules patterns` CLI command. The output of this command is a list of pattern names, as shown in the following example of command output. The keyword that is used in the rule is the Pattern from this table.

```
PTS> show policy attack rules patterns
Id Pattern                      Malware
-- ---------------------------- ------------------------
 1 alvgus                       The Alvgus 2000 Trojan
 2 amanda                       Amanda Trojan
 3 aol-admin                    AOL Admin Trojan
 4 backage                      Backage Trojan
 5 backconstruction-commands    BackConstruction Trojan
   …
```

The parameters are:

pattern The name of the pattern to match from the pattern dictionary. Use the keyword all to enable all defined patterns.

enable Set to on to enable the rule, off to disable it. Note that this is not supported if an individual IP address is specified.

This example implements the Sandvine recommended detection for the Nachia worm.

```
detection-config 1 {signature-match pattern nachia enable on}
```

18.4.8 Signature Match Auto Block

Use the signature-match-auto-block action to block specific flows.

Any flow that matches the filters defined in an enabled rule is blocked.

The syntax is:

```
detection-config <rule-number> {signature-match-auto-block
 class all|all-internal|all-external|string
 pattern all|<string> enable on|off}
```

The parameters are:

class Specifies the network class that is examined to determine if it is under attack. Options are:

- all: all network classes. The default.
- all-internal: internal network classes only.
- all-external: external network classes only.
- string: a single network class or policy class name from the subnets.txt file.

Default is all.

pattern The name of the pattern to match from the pattern dictionary. Use the keyword all to enable all defined patterns.

enable Set to on to enable the rule, off to disable it. Note that this is not supported if an individual IP address is specified.

This policy example blocks all flows which match any enabled static signature where the malicious host is from the all-external network class.

```
detection-config 1 {signature-match-auto-block class \
 all-external pattern all enable on}
```

This example blocks all flows which match any enabled static signature where the malicious host is from the any network class, except for flows where the malicious host is from an internal network class and the static signature is blaster-tftp.

```
detection-config 2 {signature-match-auto-block class all \
 pattern all enable on}
detection-config 3 {signature-match-auto-block class \
 all-internal pattern blaster-tftp enable off}
```

18.4.9 Spam

Spam detection relies upon two types of thresholds: absolute thresholds and ratio thresholds.

Absolute thresholds compare the total number of actual values observed with the threshold value. Ratio thresholds compare the total number of actual values observed over time or over another metric.

Each metric has a corresponding weight value. A host is said to be spamming if the total weight of all exceeded absolute threshold metrics is greater than the minimum absolute threshold weight or if the weight of all exceeded ratio thresholds is greater than the minimum ratio weight.

The spam rule forms are:

- General rule is used to enable/disable spam and to set general configuration parameters.
- Absolute threshold rule is used for setting threshold values.
- Ratio threshold rule is used for setting threshold values.
- Email address matching rule allows for extracting sender and recipient email addresses from SMTP traffic to match against specific email addresses, domains or domain.txt groups.

A subscriber is 'active' if he or she has sent more emails in one period than min-messages-per-period and sent email to more recipients in one period than min-recipients-per-period.

18.4.9.1 Recommended Configuration

This rule enables spam detection for all internal network classes with default threshold values.

```
detection-config 1 {spam class all-internal enable on \
 absolute-threshold-metrics on \
 ratio-threshold-metrics on}
```

Spam detection should only be used for internal subscribers.

18.4.9.2 General Rule

The general rule is used to enable or disable spam detection and for setting configuration parameters.

The syntax is:

```
detection-config <rule-number> {spam
 {class all|all-internal|all-external|string}
 {min-messages-per-period <n>}
 {min-recipients-per-period <n>}
 {enable on|off}
 {absolute-threshold-metrics on|off}
 {ratio-threshold-metrics on|off}
 {email-address-matching on|off}}
```

The parameters are:

class — Specifies the network class that is examined to determine if it is under attack. Options are:

- all: all network classes. The default.
- all-internal: internal network classes only.
- all-external: external network classes only.
- string: a single network class or policy class name from the subnets.txt file.

Default is all.

min-messages-per-period — Minimum number of email messages a subscriber must have attempted in one period to be considered active.

Default is 5, allowable range is 0 - 50000.

min-recipients-per-period — Minimum number of email recipients a subscriber must have attempted to send email to in a period to be considered active.

Default is 5, allowable range is 0 - 50000.

absolute-threshold-metrics — Set to on to enable the absolute threshold rule, off to disable it.

enable	Set to on to enable the rule, off to disable it. Note that this is not supported if an individual IP address is specified.
ratio-threshold-metrics	Set to on to enable the ratio threshold rule, off to disable it.
email-address-matching	Set to on to enable the email address matching rule, off to disable it.

This rule enables spam detection for all source classes using the default values for min-messages-per-period and min-recipients-per-period. All of the threshold values use the defaults.

```
detection-config 1 {spam class all-internal enable on \
 absolute-threshold-metrics on \
 ratio-threshold-metrics on}
```

A subscriber is 'active' if they have sent more emails in one period than min-messages-per-period and sent email to more recipients in one period than min-recipients-per-period. With the following rule in place, a subscriber must send email to more than 25 recipients and must send more then 20 email messages in one period to be considered active.

```
detection-config 2 {spam min-recipients-per-period 25 \
 min-messages-per-period 20}
```

Spam detection uses absolute metrics and ratio metrics. Absolute metrics are compared to values accumulated over the previous one hour. Ratio metrics are compared to values accumulated over the last period (either compared to time or to another metric).This rule disables absolute metrics and enables ratio metrics for all subscribers in the business network class.

```
detection-config 3 {spam class business \
 absolute-threshold-metrics off ratio-threshold-metrics   on}
```

The above three rules can be combined into a single rule, like this:

```
detection-config 4 {spam class all-internal enable on \
 min-recipients-per-period 25 min-messages-per-period 20 \
 ratio-threshold-metrics on absolute-threshold-metrics on}
```

18.4.9.3 Absolute Threshold Rule

Use the absolute threshold rule to set absolute threshold values for spam detection.

The syntax is:

```
detection-config <rule-number> {spam-absolute-thresholds
 {class all|all-internal|all-external|string}
 {min-abs-threshold-weight <n>} {unique-ehlo-name <n>}
 {unique-recipient-address <n>}
 {unique-recipient-address-weight <n>}
 {total-recipient-address <n>}
 {total-recipient-address-weight <n>}
 {unique-recipient-domain <n>}
 {unique-recipient-domain-weight <n>}
 {unique-sender-address <n>}
 {unique-sender-address-weight <n>}
 {unique-sender-domain <n>}
 {unique-sender-domain-weight <n>}
 {sessions <n>} {sessions-weight <n>}
 {attempted-messges <n>} {attempted-messages-weight <n>}
 {errors <n>} {errors-weight <n>}
 {resets <n>} {resets-weight <n>}
 {unique-servers <n>} {unique-servers-weight <n>}
 {sender-email-address-matches <n>}
 {sender-email-address-matches-weight <n>}
 {recipient-email-address-matches <n>}
 {recipient-email-address-matches-weight <n>}}
```

The parameters are:

class	Specifies the network class that is examined to determine if it is under attack. Options are:

	• all: all network classes. The default. • all-internal: internal network classes only. • all-external: external network classes only. • string: a single network class or policy class name from the subnets.txt file. Default is all.
sessions	Total number of SMTP sessions a subscriber has initiated with an SMTP server. Allowable range is 0 - 50000, default is 1000.
sessions-weight	Weight assigned to sessions. Allowable range is 0 - 10, default is 10.
total-recipient-address	Total number of recipients a subscriber has attempted to send email to. Allowable range is 0 - 50000, default is 1000.
total-recipient-address-weight	Weight assigned to total-recipient-address Allowable range is 0 - 10, default is 10.
unique-ehlo-name	Total number of unique connection names or IP addresses a subscriber uses when connecting to an SMTP server. Allowable range is 0 - 100000, default is 15.
unique-ehlo-name-weight	Weight assigned to unique-ehlo name value. Allowable range is 0 - 10, default is 10.
unique-recipient-address	Total number of unique recipients a subscriber has attempted to send email to. Allowable range is 0 - 100000, default is 100.
unique-recipient-address-weight	Weight assigned to the unique-recipients value. Allowable range is 0 - 10, default is 10.
unique-recipient-domain	Total number of unique recipient domains a subscriber has attempted to send email to. Allowable range is 0 - 100000, default is 100.
unique-recipient-domain-weight	Weight assigned to total-recipient-domains. Allowable range is 0 - 10, default is 10.
unique-sender-address	Total number of unique sender email addresses a subscriber has attempted to send email from. Allowable range is 0 - 100000, default is 30.
unique-sender-address-weight	Weight assigned to unique-send-address. Allowable range is 0 - 10, default is 10.
unique-sender-domain	Total number of unique sender email domains a subscriber has attempted to send email from. Allowable range is 0 - 100000, default is 15.
unique-sender-domain-weight	Weight assigned to unique-sender-domains. Allowable range is 0 - 10, default is 10.
unique-servers	Total number of unique SMTP servers a subscriber has connected to.

	Allowable range is 0 - 100000, default is 25.
unique-servers-weight	Weight assigned to unique-servers.
	Allowable range is 0 - 10, default is 10.
sender-email-address-matches	Total number of times a sender email address has been matched.
	Allowable range is 0-50000, default is 0.
sender-email-address-matches- weight	Weight assigned to sender-email-address-matches
	Allowable range is 0 - 10, default is 10.
recipient-email-address-matches	Total number of times a recipient email address has been matched.
	Allowable range is 0-50000, default is 0.
recipient-email-address-matches-weight	Weight assigned to recipient-email-address-matches.
	Allowable range is 0 - 10, default is 10.

Any subscriber who connects to more than 12 SMTP servers in a one hour period, will be detected as a spammer. This rule can be used to ensure internal subscribers only send email through one of 12 SMTP servers within a network.

```
detection-config 1 {spam-absolute-thresholds class \
 all-internal unique-servers 12}
```

It is possible to set different thresholds for different network classes. The following rule will set attempted-messages to 1000, unique-sender-domains to 15, and unique-recipients to 150 for all internal network classes. The second rule will override settings from the first rule, but only for the specific network class called business. In this way business subscribers can have different thresholds than non-business subscribers.

```
detection-config 2 {spam-absolute-thresholds \
 class all-internal \
 attempted-messages 1000 \
 unique-sender-domain 15 \
 unique-recipient-address 150}
detection-config 3 {spam-absolute-thresholds
 class business \
 attempted-messages 5000 \
 unique-sender-domain 10 \
 unique-recipient-address 1000}
```

Each threshold has a corresponding weight. The total weights of all exceeded thresholds must be greater than or equal to min-abs-threshold-weight before spam detects a subscriber as a spammer. With the following rule, a subscriber must exceed at least two of the absolute threshold metrics before being detected as a spammer or the subscriber must exceed unique-recipient-address or unique-sender-addresses.

```
detection-config 4 {spam-absolute-thresholds \
 class all-internal \
 unique-recipient-address 150 \
 unique-sender-address 25 \
 unique-recipient-address-weight 10 \
 unique-sender-address-weight 10 \
 min-abs-threshold-weight 20}
```

18.4.9.4 Ratio Thresholds Rule

Use the ratio threshold rule to configure threshold values.

The syntax is:

```
detection-config <rule-number> {spam-ratio-thresholds
 {class all|all-internal|all-external|<string>}
 {min-ratio-threshold-weight <n>}
 {periods-above-ratio <n>}
```

```
{periods-below-ratio <n>}
{recipients-per-period <n>} {recipients-per-period-weight <n>}
{unique-senders-per-period <n>}
{unique-senders-per-period-weight <n>}
{attempted-messages-per-period <n>}
{attempted-messages-per-period-weight <n>}
{sessions-per-period <n>} {sessions-per-period-weight <n>}
{servers-per-period <n>} {servers-per-period-weight <n>}
{attempted-messages-per-successful-message <n>}
{attempted-messages-per-successful-message-weight <n>}}
```

The syntax is:

class

Specifies the network class that is examined to determine if it is under attack. Options are:

- all: all network classes. The default.
- all-internal: internal network classes only.
- all-external: external network classes only.
- string: a single network class or policy class name from the subnets.txt file.

Default is all.

attempted-messages-per-period

Total number of email messages a subscriber has attempted to send per minute.

Allowed range is 0 - 50000, default is 25.

attempted-messages-per-period- weight

Weight assigned to attempted-messages-per-period.

Allowed range is 0 - 10, default is 10.

attempted-messages-per-successful-message

Total number of email messages a subscriber has attempted to send for every successfully s

Allowed range is 0 - 50000, default is 10.

attempted-messages-per-successful-message-weight

Weight assigned to attempted-messages-per- successful-msgs.

Allowed range is 0 - 10, default is 10.

min-ratio-threshold-weight

The minimum combined weight of all exceeded ratio threshold metrics before a subscriber may be considered a spammer.

Allowed range is 0 - 80, default is 10.

periods-above-ratio

The minimum number of consecutive periods a subscriber must be above min-ratio-threshold-weight before being considered a spammer

Allowed range is 0 - 100000, default is 2.

periods-below-ratio

The minimum number of consecutive periods a subscriber must be below min-ratio-threshold-weight to no longer be considered a spammer.

Allowed range is 0 - 100000, default is 2.

recipients-per-period

Total number of recipient email addresses a subscriber has attempted to send email to per period (minutes).

Allowed range is 0 - 50000, default is 15.

recipients-per-period-weight

Weight assigned to recipients-per-period.

Allowed range is 0 - 10, default is 10.

servers-per-period	Total number of unique SMTP servers a subscriber has connected to per period (minutes). Allowed range is 0 - 100000, default is 5.
servers-per-period-weight	Weight assigned to servers-per-period. Allowed range is 0 - 10, default is 10.
sessions-per-period	Total number of SMTP sessions a subscriber has initiated with an SMTP server per period (minutes). Allowed range is 0 - 50000, default is 25.
sessions-per-period-weight	Weight assigned to sessions-per-period. Allowed range is 0 - 10, default is 10.
unique-senders-per-period	Total number of unique sender email addresses a subscriber has attempted to send email to per period (in minutes). Allowed range is 0 - 100000, default is 10.
unique-senders-per-period-weight	Weight assigned to unique-senders-per-period. Allowed range is 0 - 100000, default is 10.

To help filter subscribers who are not spamming, but are on dial-up and bursting email, ratio threshold rule can be configured such that a subscriber must exceed ratio thresholds for more than one consecutive period. With this rule, a subscriber is not detected as a spammer until they have exceeded ratio thresholds for 4 periods consecutively (20 minutes). The subscriber is no longer considered sending spam when they have dropped below ratio thresholds for two consecutive periods (10 minutes).

```
detection-config 1 {spam-ratio-thresholds \
 class all-internal \
 periods-above-ratio 4 periods-below-ratio 2}
```

18.4.9.5 Email Address Matching Rule

Use the email address matching feature to extract sender and recipient email addresses from the SMTP traffic and match them against particular sections (groups) from domain.txt

You can filter specific email addresses, filter all email addresses under a domain, or filter all email addresses under subdomains of a domain. Address matches trigger increments in absolute thresholds.

The syntax is:

```
detection-config <rule-number> {spam-email-address-matching
 {class all|internal|all-external|<string>}
 {sender-list-groups <string>}
 {recipient-list-groups <string>}}
```

The parameters are;

class	Specifies the network class that is examined to determine if it is under attack. Options are: • all: all network classes. The default. • all-internal: internal network classes only. • all-external: external network classes only. • string: a single network class or policy class name from the subnets.txt file. Default is all.
sender-list-group-string	Comma delimited list of groups from domains.txt to match sender email addresses against.
recipient-list-groups	Comma delimited list of groups from domains.txt to match recipient email addresses against.

For detailed information on creating a domains.txt file, see *Domain List File* on page 20.

Assuming that the domain.txt file contains groups groupA, groupB, groupC, the following rule enables email address matching with sender addresses being matched against groupA and groupB and recipient addresses being matched against groupB and groupC:

```
detection-config 1 {spam email-address-matching class all on }
detection-config 2{spam-email-address-matching class all \
 sender-list-groups "groupA, groupB" \
 recipient-list-groups "groupB, groupC" }
```

To actually make the above rules useful, absolute thresholds must be enabled and configured as in this example.

```
detection-config 3 { spam-absolute-thresholds class all \
 sender-email-address-matches 0 \
 sender-email-address-matches-weight 10 \
 recipient-email-address-matches 0 \
 recipient-email-address-matches-weigth 10 }
```

The above rule will trigger the address match thresholds if at least one sender or recipient address is matched for a particular host.

18.4.10 DNS Domain Flood

Use the domain flood rule to detect when DNS queries to a particular domain in a domain-group (defined below) exceeds a threshold.

It can also be used to mitigate or block requests to a particular domain. For malicious domains where all queries should be blocked, the threshold should be set to 0. The specified domains come from the domains.txt file (see *Domain List File* on page 20).

This rule can be configured to only detect certain query request types, such as type A or MX. The syntax is:

```
detection-config <rule-number> {dns-domain-flood\
 {class all|all-internal|all-external|<string>}\
 {request-type all|A|MX|CNAME|PTR|OTHER|AGGREGATE}\
 {domain-group <string>} {threshold <n>} {period <n>}\
 {auto-block on|off} {enable on|off}}
```

When there are multiple enabled detections for the same class with different detection periods, the smaller detection period will be used. Class implies the network/policy class that applies to DNS clients.

The parameters are:

class Specifies the network class that is examined to determine if it is under attack. Options are:

- all: all network classes. The default.
- all-internal: internal network classes only.
- all-external: external network classes only.
- string: a single network class or policy class name from the subnets.txt file.

Default is all.

request-type Specifies the request types that the rule will apply to. Options are:

- all: all request types, not including AGGREGATE.
- AGGREGATE request-type is not a DNS request type; rather it means the sum of all requests irrespective of the type.
- A: Address record
- MX: Mail Exchange

- CNAME: Canonical Name
- PTR: Pointer record
- OTHER: records classified as other such as service (SVR)

The default is all.

domain-group A single domain group from domains.txt. Default is all.

threshold Detection threshold expressed as DNS requests per second. For a user to be identified as malicious they must exceed the threshold. If the threshold is 10 requests per second, then the user must send at least 101 requests during the 10 second interval.

Allowed range is 0 - 2184 seconds requests per second. Default is 2184.

period Time over which the analysis is performed. This starts whenever the network protection policy is read. Specified as: `s|seconds m|minutes h|hours d|days`. For example: `10s 30seconds 6h 12hours`

Allowable range is 1-30, default is 30seconds.

auto-block Set to on to block hosts that exceed the threshold during the period for the duration of the period. Set to off to disable blocking hosts. Default is off.

enable Set to on to enable the rule, off to disable it. Note that this is not supported if an individual IP address is specified.

This example blocks all requests to all types of malicious domains. Since the threshold is 0, and auto-block is "on", all requests to any domain in the bad_domains domain group as defined in the domains.txt file will be blocked.

```
detection-config 1 {dns-domain-flood threshold 0 enable on \
 domain-group bad_domains auto-block on}
```

This example detects when a large number of DNS queries are made to a particular domain group over a short amount of time, such as 5 seconds, irrespective of the request type.

```
detection-config 2 {dns-domain-flood request-type AGGREGATE \
 threshold 200 period 5 enable on domain-group \
 protected_domains}
```

This example identifies when users from a particular network class are generating DNS queries to a specified domain group, using different thresholds for different request types.

```
detection-config 3 {dns-domain-flood class subs \
 request-type A threshold 5 period 1 enable on \
 domain-group interesting_domains}
detection-config 4 {dns-domain-flood class subs request-type \
 CNAME threshold 10 period 1 enable on domain-group \
 interesting_domains}
```

This example uses WMD rules to block packets from hosts generating an average of 200 DNS request queries per second, irrespective of the request type, to the protected_domains domain group over a 5 second interval. Packets from the host are blocked for 10 seconds.

```
detection-config 5 {dns-domain-flood request-type AGGREGATE \
 threshold 200 period 5 enable on domain-group \
 protected_domains}
aggregator-config 6 {dns-domain-flood sample-period 0 \
 detection-timeout 10}
wmd-rule 7 {dns-domain-flood class all} {limit-host-packets \
 transport all}
```

18.4.11 DNS Request Flood

Use DNS request flood to detect when DNS queries of a particular type exceed a threshold.

You can also use this type of rule to mitigate or block requests of a particular type.

The syntax is:

```
detection-config <rule-number> {dns-request-flood \
 {class all|all-internal|all-external|<string>}\
 {request-type all|A|MX|CNAME|PTR|OTHER|AGGREGATE}\
 {threshold <n>} {period <n>} {auto-block on|off}\
 {enable on|off}}
```

Class implies the network/policy class that applies to DNS clients. When there are multiple enabled detections for the same class with different detection periods, the smallest detection period is used.

The parameters are:

class Specifies the network class that is examined to determine if it is under attack. Options are:

- all: all network classes. The default.
- all-internal: internal network classes only.
- all-external: external network classes only.
- string: a single network class or policy class name from the subnets.txt file.

Default is all.

request-type Specifies the request types that the rule will apply to. Options are:

- all: all request types, not including AGGREGATE.
- AGGREGATE request-type is not a DNS request type; rather it means the sum of all requests irrespective of the type.
- A: Address record
- MX: Mail Exchange
- CNAME: Canonical Name
- PTR: Pointer record
- OTHER: records classified as other such as service (SVR)

The default is all.

threshold Detection threshold expressed as DNS requests per second. For a user to be identified as malicious they must exceed the threshold. If the threshold is 10 requests per second, then the user must send at least 101 requests during the 10 second interval.

Allowed range is 0 - 2184 seconds requests per second. Default is 2184.

period Time over which the analysis is performed. This starts whenever the network protection policy is read. Specified as: `s|seconds m|minutes h|hours d|days`. For example: `10s 30seconds 6h 12hours`

Allowable range is 1-30, default is 30seconds.

auto-block Set to on to block hosts that exceed the threshold during the period for the duration of the period. Set to off to disable blocking hosts. Default is off.

enable Set to on to enable the rule, off to disable it. Note that this is not supported if an individual IP address is specified.

This rule sets the same threshold for all request types. It detects when a host in the all-internal network class sends more than 5 requests per second during a 5 second interval of the same request-type. For example, 25 type A requests or 25 type MX requests.

```
detection-config 1 {dns-request-flood class all-internal \
 request-type all threshold 5 period 5 enable on}
```

This example is similar to the previous one, except that in this case, a threshold is set for the total of all requests irrespective of the request type (request-type of AGGREGATE).

```
detection-config 2 {dns-request-flood class all-internal \
 request-type AGGREGATE threshold 5 period 5 enable on}
```

This rule prevents subscribers from making too many requests of a specific type by blocking requests after a threshold is exceeded.

```
detection-config 3 {dns-request-flood class all-internal \
 request-type A threshold 50 period 1 auto-block on enable on}
```

This example sets different thresholds for different request types.

```
detection-config 4 {dns-request-flood class all-internal \
 request-type A threshold 50 period 1 enable on}
detection-config 5 {dns-request-flood class all-internal \
 request-type MX threshold 25 period 1 enable on}
detection-config 6 {dns-request-flood class all-internal \
 request-type OTHER threshold 100 period 1 enable on}
```

This example uses WMD rules to block packets from hosts generating an average of 200 DNS request queries per second, irrespective of the request type, over a 5 second interval. Packets from the host are blocked for 10 seconds.

```
detection-config 7 {dns-request-flood request-type \
 AGGREGATE  threshold 200 period 5 enable on}
aggregator-config 8 {dns-request-flood sample-period 0 \
 detection-timeout 10}
wmd-rule 9 {dns-request-flood class all} \
 {limit-host-packets  transport all}
```

18.4.12 DNS Outstanding Sessions

Use the DNS outstanding sessions rule to detect and limit the number of active sessions, where a session is a unique DNS transaction, between a client and a DNS server.

The syntax is:

```
detection-config <rule-number> {dns-outstanding-sessions \
 {class all|all-internal|all-external|<string>}\
 {request-type all|A|MX|CNAME|PTR|OTHER|AGGREGATE}\
 {threshold <n>} {auto-block on|off} {enable on|off}}
```

Note that:

- Class implies the network/policy class that applies to DNS clients.
- A single detection period is used for all the dns-outstanding-sessions detection-config rules.
- The detection period default is 30 seconds. The minimum permitted is 1 second and the maximum is 30 seconds.

To change the detection period, use the `set config policy network-protection dns outstanding-sessions-period` CLI command.

The parameters are:

class Specifies the network class that is examined to determine if it is under attack. Options are:

- all: all network classes. The default.
- all-internal: internal network classes only.
- all-external: external network classes only.
- string: a single network class or policy class name from the subnets.txt file.

Default is all.

request-type Specifies the request types that the rule will apply to. Options are:

- all: all request types, not including AGGREGATE.
- AGGREGATE request-type is not a DNS request type; rather it means the sum of all requests irrespective of the type.
- A: Address record
- MX: Mail Exchange
- CNAME: Canonical Name
- PTR: Pointer record
- OTHER: records classified as other such as service (SVR)

The default is all.

request-type Specifies the request types that the rule will apply to. Options are:

- all: all request types, not including AGGREGATE.
- AGGREGATE request-type is not a DNS request type; rather it means the sum of all requests irrespective of the type.
- A: Address record
- MX: Mail Exchange
- CNAME: Canonical Name
- PTR: Pointer record
- OTHER: records classified as other such as service (SVR)

The default is all.

threshold Gauge as to the number of outstanding sessions.

Allowed range is 1 - 65534 seconds requests per second. Default is 65534.

auto-block Set to on to block hosts that exceed the threshold during the period for the duration of the period. Set to off to disable blocking hosts. Default is off.

enable Set to on to enable the rule, off to disable it. Note that this is not supported if an individual IP address is specified.

This example detects when any host from all internal network classes has more than five outstanding sessions of request type A.

```
detection-config 1 {dns-outstanding-sessions class all-internal \
 request-type A threshold 5 enable on}
```

This example detects and limits the number of outstanding sessions of all types to ten.

```
detection-config 2 {dns-outstanding-sessions class all \
 request-type all threshold 10 auto-block on enable on}
```

This example detects and limits the number of outstanding sessions, irrespective of type, to 20.

```
detection-config 3 {dns-outstanding-sessions class all \
 request-type AGGREGATE threshold 20 auto-block on enable on}
```

This example uses WMD rules to block packets from hosts with more than 20 outstanding DNS sessions, irrespective of the request type. Packets from the host will be blocked for 10 seconds.

```
detection-config 4 {dns-outstanding-sessions request-type \
 AGGREGATE threshold 20 enable on}
aggregator-config 5 {dns-outstanding-sessions \
 sample-period 0 detection-timeout 10}
wmd-rule 6 {dns-outstanding-sessions class all} \
 {limit-host-packets transport all}
```

18.4.13 Overlapping Configuration

Each of the malicious traffic detections provided by network protection may be tailored, if necessary, to suit a given situation.

The tailoring takes a hierarchical form, such that settings applying to the general situation may be specified once, whereas settings applying to specific situations may be specified as necessary. This arrangement allows complex sets of detection scenarios to be created using only a small set of rules.

The detection types Address scan, flow flood, SYN flood, user bandwidth, static signature and spam each supply one or more “filters” that may be used to tailor them:

```
Source Class (src-class)
Destination Class (dst-class)
Class (network or policy class)
Network Protocol (network)
Transport Protocol (transport)
IP Address (ip-address)
IP Port (port)
Total Recipients (value)
Pattern (static signature name)
```

The filters listed above may be used alone or in combination to "target" a set of parameters to a specific set of network traffic. Consider these address scan rules:

```
detection-config 1 {address-scan class all period 30}
```

This rule sets the address scan detection period for address scans originating from all cost-classes to be 30 seconds:

```
detection-config 2 {address-scan class all-internal 40}
```

This rule will set the address scan detection period for address scans originating from internal cost-classes to be 40 seconds. When considered together, these rules mean:

```
set the period for address scans for users in internal cost-classes to 40 seconds and all others
 to 30 seconds.
```

Filter settings from one rule may be "overridden" by settings in a different rule. Thus, a small set of "default" rules may be created, followed by another set of "more specific" rules for special cases. A more specific set of filters always take precedence over a less specific set. Consider these rules:

1. This rule sets the threshold for all address scans to be 20 flows per second. Note: any filter that is not specified means “all”.

   ```
   detection-config 3 {address-scan threshold 20}
   ```

2. This rule sets the threshold for internal address scans to be 15 flows per second.

   ```
   detection-config 4 {address-scan class all-internal \ threshold 15}
   ```

3. This rule sets the threshold for tcp flows to be 25 flows per second.

```
detection-config 5 {address-scan transport tcp threshold 25}
```

4. This rule sets the threshold for flows destined to port 80 to be 40 flows per second.

```
detection-config 6 {address-scan port 80 threshold 40}
```

To fully understand what this set of rules evaluates to, it is essential to enumerate the possibilities for each of the filters:

- class: internal and external (all means both)
- transport: tcp, udp and icmp (all means all three)
- port: * (not specified) or 80 (in the example).

By listing all permutations of these values we can see what value “threshold” takes on for each combination:

class	transport	port	threshold	rule
internal	tcp	80	40	6
internal	tcp	*	25	5
internal	udp	80	40	6
internal	udp	*	15	4
internal	icmp	80	40	6
internal	icmp	*	15	4
external	tcp	80	40	6
external	tcp	*	25	5
external	udp	80	40	6
external	udp	*	20	3
external	icmp	80	40	6
external	icmp	*	20	3

A single rule may specify multiple filter settings:

```
detection-config 1 {address-scan class all-internal \
  transport tcp port 135 threshold 200}
```

This rule will affect only address scan detection of tcp flows on port 135 from internal users.

Each of the detection types specify one or more parameters that affect the detection being performed. The definition of the parameters is quite independent and each may be specified with a set of rules quite separately from the others:

1. address-scan class all period 30
2. address-scan transport tcp period 40
3. address-scan threshold 15
4. address-scan class all-internal threshold 20
5. address-scan transport tcp port 80 threshold 30

class	transport	port	period	threshold	rule
internal	tcp	80	40	30	2,5

class	transport	port	period	threshold	rule
internal	tcp	*	40	20	2,4
internal	udp	80	30	20	1,4
internal	udp	*	30	20	1,4
internal	icmp	80	30	20	1,4
internal	icmp	*	30	20	1,4
external	tcp	80	40	30	2,5
external	tcp	*	40	15	2,3
external	udp	80	30	15	1,3
external	udp	*	30	15	1,3
external	icmp	80	30	15	1,3
external	icmp	*	30	15	1,3

Each of the detection rules specifies an additional parameter (enable) which is used to actually "turn on" detections using the parameters specified in the rules. The concept here is that a set of rules will be used to set up some typical values for detections of the various types (without enabling any of them). Additional rules may then be created to tailor detections to very specific sets of traffic (perhaps not enabling them either). Then, a small set of rules is created to actually enable the detection of malicious traffic on only the sections of traffic desired.

18.5 Network Protection Aggregator Configuration (aggregator-config)

The network protection aggregator collects input events over a period of time to generate new output events, trigger actions, and generate a periodic summary statistical information.

This enhances the ability of network protection to detect network anomalies which may indicate worm activity. Aggregation also reduces the number of false positives that are detected by:

- Reducing the incidence of hosts being detected at the source or destination of malicious internet traffic.
- Reducing the number of statistic records being written to the database.
- Sending notifications on the detection of significant host activities.
- Detecting and reporting on the activities of groups of hosts.

The aggregator generates statistics, which network operators may use to monitor the quality of service and measure bandwidth use for planning purposes. Other statistics indicate the amounts of malicious traffic flowing in the network as well as the amount of traffic affected by the various mitigations. This data can be applied to hosts which the aggregator has determined to be the source or destination of this kind of traffic.

The aggregator may call upon the services of the available mitigating plug-ins to forcibly make changes to the traffic patterns of a specific host, or group of hosts, to reduce the damage caused by that traffic and preserve the quality of service for the remaining subscriber base.

Policy rules may include threshold values to control when a mitigating action against a host is applied and when it is removed. You can use timers to provide filtering effects to minimize oscillating behavior when traffic statistics vary close to the configured threshold values.

The output of the aggregator provides:

- raw information for demographic reports
- ability to control mitigation, block, reshape, direct to a captive portal and so forth
- notification by email.

18.5.1 Host Grouping

The default and most common host grouping is the timed-host grouping. Which means that in this detection rule's case, the group consists of a single timed-host, a detection that we keep state on and time a single host for malicious activity.

The second grouping is an all-hosts grouping which groups all of the timed-host detections (for all hosts over a detection type matching the given filters) and keeps state and aggregate statistics on all of them. This is very useful if you wish to take mitigation action when the number of hosts detected for a specific detection breach a threshold, or their combined packet-rate crosses a threshold, or their combined bit-rate passes a percentage threshold of the total bandwidth of your link. In these situations, an email-alert action is typically taken to notify a network administrator of abnormally high malicious activity.

18.5.2 Relationship between Events and Mitigating Actions

An event is the observation of a detection based on traffic patterns and data. Network protection takes action to mitigate the event.

An event may or may not be malicious and a host may be either the attacker or under an attack. Once the host is identified as attacking or under attack, this designation remains until the detection session is terminated.

18.5.2.1 Detection Session for Timed Host Group

A timed host group initially collects one entire sample period of events to determine if an attack is in progress.

This reduces the probability of false positives. Once an event is identified, each subsequent event restarts the sample timer. When an entire sample time has passed without any events being received, the detection session ends.

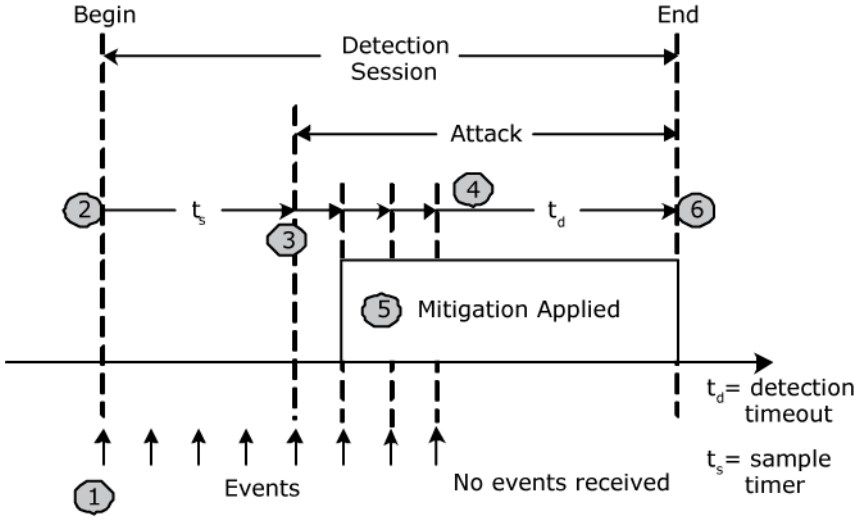


Figure: Timed host group detection session

1. A detection session is created when the first event is received for a host. Note the first event does not indicate an attack.
2. The host sample time (ts) is started.
3. When the sample timer (ts) expires, the amount of time the host was actively attacking is compared to the sample period. If the percent time the host was active exceeds the configure threshold, the detection is marked as an attack.

   The host sample timer is restarted on each event received using the value calculated as the time difference between the oldest and second oldest host event samples.
4. The detection timeout timer (td) is started when the first event is received and is restarted on each event received.
5. Mitigating actions may or may not be applied during the lifetime of the attack.
6. When the detection timeout timer expires, all applied actions are removed and the detection session is destroyed.

18.5.2.2 Sampled Host Timing

In this example, the host sample timer is started and the sample timer expires (points 2 and 3 on the previous illustration).

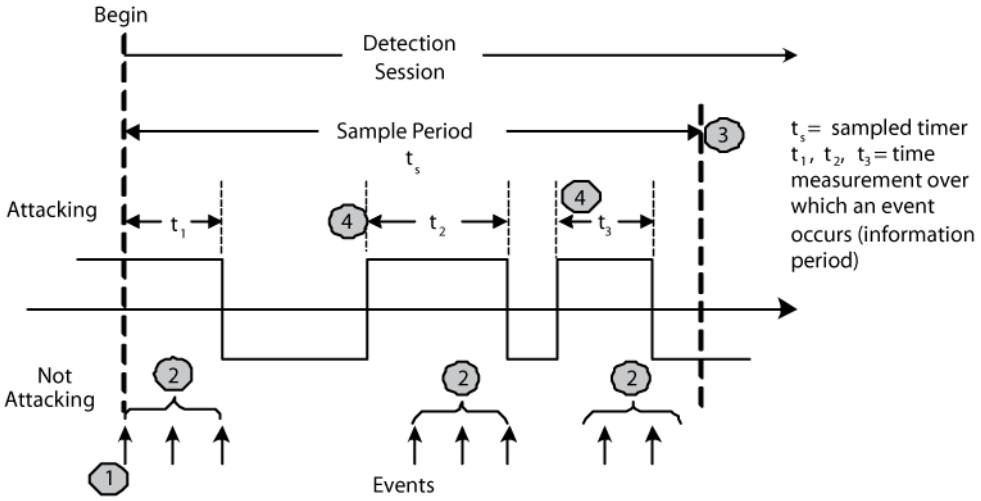


Figure: Sampled host timing

1. A detection session is created when the first event is received for a host. Note that the first event does not indicate an attack. The host sample timer (ts) is started.
2. Events are collected while the sample timer is running.
3. On expiry of the sample timer the aggregator will sum the total time the host was attacking and calculate a percent value of the attacking time out of the sample period. If this value exceeds a configured threshold, the host is marked as attacking.

   ((t1 +t2 +t3)/ts) *100 = percentage of time host was involved in malicious activity. This must break a threshold value (timed_hosts_percent) to signal the beginning of an attack.
4. The sample timer is restarted using the calculated time difference between the oldest and second oldest events in the sample set. The sample window is moved along by restarting the sample timer with the calculated time difference between the oldest and second oldest events. The oldest event will be dropped from the event sample set at that time. The percent calculation is performed on each sample timeout event.

18.5.2.3 Remove Action Delay

The remove action delay extends the application of mitigating actions to prevent oscillation between removing/applying actions for attack behaviors which are close to the configured thresholds.

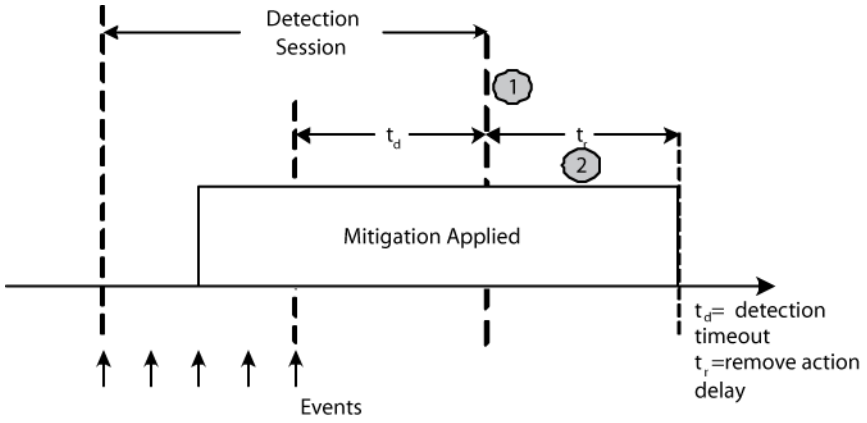


Figure: Remove action delay

1. In this example, the detection timeout timer expired with no events received in the entire sample period. The aggregator marks the end of the attack and begins to remove the applied mitigation actions.

   If the rule doesn't specify a remove action delay timeout, the actions will be removed immediately 'as the detection session is destroyed.

2. If the rule specifies a remove action delay timeout, when the detection session is destroyed, the remove action delay timer is started. At this point, the action is guaranteed to be present for exactly the length of the timeout.

   Events received after the detection session is destroyed will result in a new detection session and a mitigation action which is independent (even if it is identical) to the old action that is in the remove action delay state.

18.5.2.4 Sample Timer and Remove Action Delay Timer Examples

In this example, the low threshold of the active rule was crossed and the aggregator began the process of removing the applied actions.

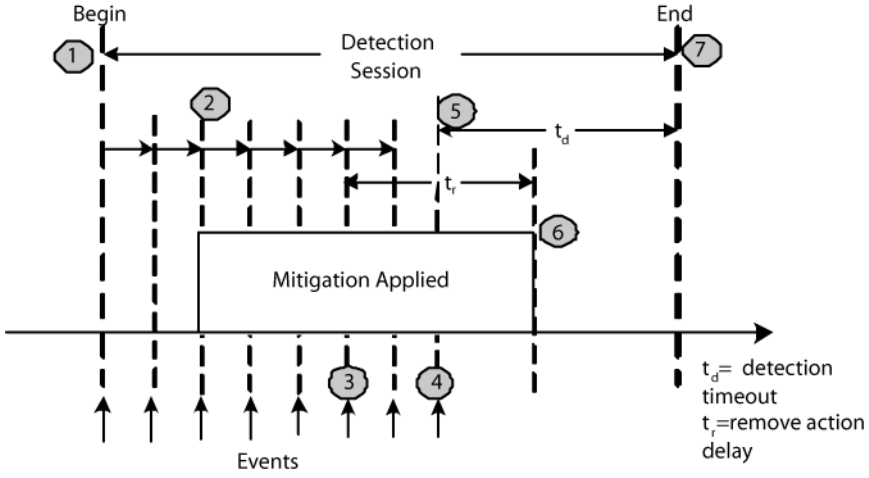


Figure: Removing applied actions

1. Detection is started when events are received.
2. Mitigation action is applied because the high threshold has been crossed (such as bit-rate, packet-rate, total-hosts and so on) and the timed host percent threshold has been crossed.
3. The aggregator starts the remove action time as specified in the rule because the low threshold is crossed.

4. Continued events to the aggregator indicates the attack is still in progress. Once the host is identified as attacking, this designation remains until the detection session is terminated; however the events do not trigger re-application of any mitigating actions while the remove action delay timer is running.
5. The events stop and the detection timeout timer is started.
6. The remove action timer, if any, expires and the applied actions are removed.
7. The detection session remains until the detection timeout timer expires at which time the detection session is destroyed.

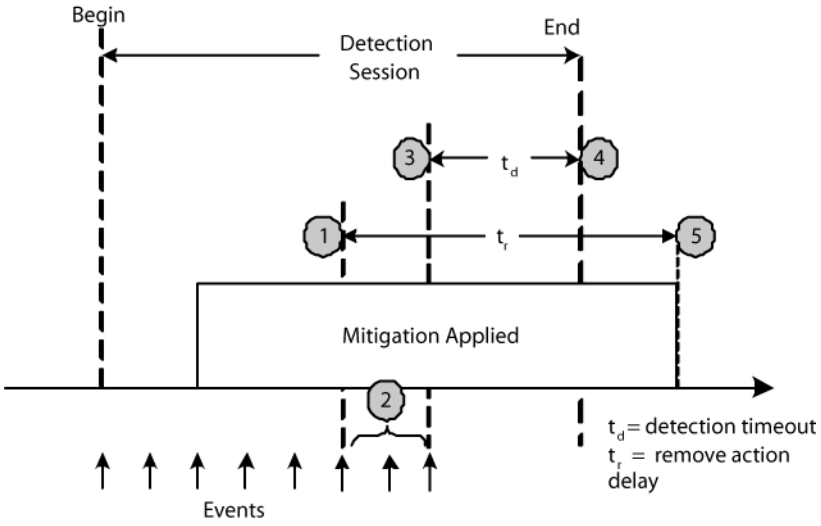


Figure: Sample timer is shorter than the remove action delay

1. In this example, the low threshold of the active rule was crossed and the aggregator began the process of removing the applied actions.

   The aggregator starts the remove action timer as specified in the rule.
2. Events are sent to the aggregator which indicates the attack is still in progress. Subsequent events do not trigger the application of mitigation actions while the remove action delay timer is running.
3. The events stop and the detection timeout timer is restarted once more.
4. The detection timeout timer expires when the remove action timer is running.
5. The remove action timer expires, the applied actions are removed.

18.5.3 Aggregator Configuration Rules

Use an aggregator configuration rule to give global operating parameters to the aggregator.

The syntax is:

```
aggregator-config <rule-number> {configuration clause}
```

An aggregator configuration rules has a single configuration clause. The configuration clause may have zero or more optional parameters following the detection name keyword. Unspecified parameters are set to their default value.

```
aggregator-config <rule-number> {detection-name sample-period <n>
 detection-timeout <n> summary-period <n> timed-hosts-percent <p>}
```

The parameters are:

detection-name Indicates the detection type.

detection-timeout Specifies the maximum amount of time the aggregator will wait since receiving the last detection event before destroying the detection session.

Valid units include: s|seconds m|minutes h|hours d|days. Default unit is seconds if none is specified. Default value is 300 seconds, which is five minutes.

sample-period Specifies the size of the sliding window which delimits the set of events to be used when filtering malicious host events. This filtering is used to prevent a host from oscillating in and out of the detection state. The aggregator remembers the set of detection events time stamped from the current time minus the sample window size. The oldest detection event is discarded as it falls out of the sample window.

Valid units include: s|seconds m|minutes h|hours d|days. Default unit is seconds if none is specified. Default value is 300 seconds, which is five minutes.

summary-period Specifies the period between intermediate (summary) statistic records for active detection sessions.

Valid units include: s|seconds m|minutes h|hours d|days. Default unit is seconds if none is specified. Default value is 900 seconds, which is 15 minutes.

timed-host-percent The events in the sample window are accumulated to calculate a percentage ratio between the total of all event periods and the sample window. The timed-host-percent parameter specifies the percentage threshold of the sample window period consumed by the sum of the event periods. If the calculated percent value is greater than this parameter value the host in question is flagged as actively involved in malicious activity: the official beginning of the attack (based on the detection name) at which point the first statistic is logged to the database and the wmd-rules are evaluated to determine the appropriate mitigation action.

Allowed range is 1-100%. Default is 40%.

18.6 Mitigation Rules

Malicious traffic mitigation actions are implemented by the CND process.

18.6.1 General Mitigation Parameters

The general wmd-rule parameters are:

flows The maximum number of flows to allow with the same destination address.

max-actions The maximum number of conditional actions implemented by the network protection module. This parameter applies to conditional actions only.

The default value for max-actions is 1000. Higher values are not recommended because each conditional action implemented has a non-linear impact on performance. Lower values for max-actions are encouraged.

order Determines the order in which actions are applied. If multiple actions from one or more rules have the same order value, it is assumed that the actions are completely independent and the actual implementation order is dependent on many factors.

Order is a value from 1 to 10. Actions assigned an order of 1 are implemented before order 2 actions and so forth.

18.6.2 Aggregator Policy Rules

Aggregator policy rules determine what the aggregator does with each specific event, such as to indicate which events are accumulated for a specific host.

If there are no aggregator policy rules for an event, the statistics or event data is written directly to the database without aggregation.

```
wmd-rule <rule-number> {detection-type
 group timed-host|all-hosts
 class all|all-internal|all-external|<string>
 {transport tcp|udp|icmp} {dst-port <n>|all} {pattern <string>}
 {bit-rate-high <n>} {bit-rate-low <n>}
 {percent-bit-rate-high <n>} {percent-bit-rate-low <n>}
 {packet-rate-high <n>} {packet-rate-low <n>} {duration <n>}
 {total-hosts-high <n>} {total-hosts-low <n>}
 {total-recipients <n>}
 {remove-action-delay <n>}}
 {action-clause1}
 {action-clause2}
 …
```

The parameters are:

class Specifies the network class that is examined to determine if it is under attack. Options are:

- all: all network classes. The default.
- all-internal: internal network classes only.
- all-external: external network classes only.
- string: a single network class or policy class name from the subnets.txt file.

Default is all.

dst-port Destination port.

Can be tpc|udp|all. Default is all.

pattern The name of the pattern to match from the pattern dictionary. Use the keyword all to enable all defined patterns.

transport Specifies the transport protocol. Can be one of: tcp|udp|icmp|all. Default is all.

detection-type Indicates the detection type.

group Instructs the aggregator on which hosts to include when processing the rule. See *Host Grouping* on page 215.

Allowed values are: timed-host|all-hosts. Default is timed-host.

bit-rate-high The maximum allowed average bitrate permitted in the event reporting period. If this threshold is exceeded mitigating actions will be applied.

Allowed values are in the range 1 - 4294967295.

bit-rate-low The minimum average bitrate required to remove any mitigating actions that have been applied.

Allowed values are in the range 1 - 4294967295.

percent-bit-rate-high The maximum allowed average bitrate represented as a percent of the total bandwidth permitted in the event reporting period. If this threshold is exceeded mitigating actions will be applied.

Allowed values are in the range 0.01 - 100.

percent-bit-rate-low The minimum average bitrate represented as a percent of the total bandwidth required to remove any mitigating actions that have been applied.

Allowed values are in the range 0 - 99.99.

packet-rate-high The maximum allowed average packet rate permitted in the event reporting period. If this threshold is exceeded mitigating actions will be applied.

Allowed values are in the range 1 - 4294967295.

packet-rate-low The minimum average packet rate required to remove any mitigating actions that have been applied.

Allowed values are in the range 1 - 4294967295.

total-hosts-high For a grouped host rule, the total number of hosts which are currently a member of the group. If this threshold is exceeded mitigating actions will be applied.

Allowed values are in the range 1 - 999999999.

total-hosts-low For a grouped host rule, the minimum number of hosts which are currently a member of the group required to remove any mitigation actions that have been applied.

Allowed values are in the range 0 - 999999999.

total-recipients Total number of email recipients to whom a subscriber has attempted to send mail. This filter is not a rate but is a cumulative statistic for the duration of the detection.

Allowed values are in the range 1 - 999999999.

duration The duration timeout represents the maximum amount of time a detection session may exist without being subject to any mitigating actions. Once it has expired the host is open to mitigating actions until the detection session is destroyed.

Allowed values are in the range 0 - 2999940. Default is 0, which means disabled.

remove-action-delay When configured, this timer instructs the aggregator to delay removing the applied mitigating actions until this period of time has passed. If during this time an event is received which results in the request that the mitigating actions be re-applied, the remove action delay timer will be stopped and the applied actions will remain in place undisturbed. If the detection sample timer expires while the remove action delay timer is running, and the aggregator did not receive any events for the host in that period, the aggregator will stop the remove action delay timer, remove all of the applied mitigating actions, and subsequently terminate the detection session.

Allowed values are in the range 0 - 99999999. Default is 0, which means disabled.

The filters and parameters that are applicable based on the detection type are:

Detection type	Filters	Filter keyword	Parameters	Parameter keyword
address-scan	Host grouping	group	Bitrate threshold	bit-rate-high bit-rate-low
	Class	class	Packet rate thresholds	packet-rate-high packet-rate-low
	Transport protocol	transport	Percent rate thresholds	percent-bit-rate-high percent-bit-rate-low
	Destination port	dst-port	Total hosts thresholds	total-hosts-high total-hosts-low
		Duration	duration	
		Remove action delay	remove-action-delay	
flow-flood	Host grouping	group	Bitrate threshold	bit-rate-high bit-rate-low
	Class	class	Packet rate thresholds	packet-rate-high packet-rate-low

Detection type	Filters	Filter keyword	Parameters	Parameter keyword
	Transport protocol	transport	Percent rate thresholds	percent-bit-rate-high percent-bit-rate-low
	Destination port	dst-port	Total hosts thresholds	total-hosts-high total-hosts-low
		Duration	duration	
		Remove action delay	remove-action-delay	
signature-match	Host grouping	group	Bitrate thresholds	bit-rate-high bit-rate-low
	Class	class	Packet rate thresholds	packet-rate-high packet-rate-low
	Pattern name	pattern	Percent rate thresholds	percent-bit-rate-high percent-bit-rate-low
	Transport protocol	transport	Total hosts thresholds	total-hosts-high total-hosts-low
	Destination port	dst-port	Duration	duration
		Remove action delay	remove-action-delay	
spam	Host grouping	group	Bitrate thresholds	bit-rate-high bit-rate-low
	Class	class	Packet rate thresholds	packet-rate-high packet-rate-low
		Percent bitrate thresholds	percent-bit-rate-high percent-bit-rate-low	
		Total hosts thresholds	total-hosts-high total-hosts-low	
		Total recipients	total-recipients	
		Duration	duration	
		Remove action delay	remove-action-delay	
syn-flood	Host grouping	group	Bitrate thresholds	bit-rate-high bit-rate-low
	Class	class	Packet rate thresholds	packet-rate-high packet-rate-low
	Destination port	dst-port	Percent bitrate thresholds	percent-bit-rate-high percent-bit-rate-low
		Total hosts thresholds	total-hosts-high total-hosts-low	
		Duration	duration	
		Remove action delay	remove-action-delay	
user-bandwidth	Host grouping	group	Bitrate thresholds	bit-rate-high bit-rate-low
	Class	class	Packet rate thresholds	packet-rate-high packet-rate-low
		Percent bitrate thresholds	percent-bit-rate-high percent-bit-rate-low	

Detection type	Filters	Filter keyword	Parameters	Parameter keyword
		Total hosts thresholds	total-hosts-high total-hosts-low	
		Duration	duration	
		Remove action delay	remove-action-delay	
dns-domain- flood	Host grouping	group	Bitrate thresholds	bit-rate-high bit-rate-low
	Class	class	Packet rate thresholds	packet-rate-high packet-rate-low
		Total hosts thresholds	total-hosts-high total-hosts-low	
		Duration	duration	
		Remove action delay	remove-action-delay	
dns-request- flood	Host grouping	group	Bitrate thresholds	bit-rate-high bit-rate-low
	Class	class	Packet rate thresholds	packet-rate-high packet-rate-low
		Total hosts thresholds	total-hosts-high total-hosts-low	
		Duration	duration	
		Remove action delay	remove-action-delay	
dns-outstanding-sessions	Host grouping	group	Bitrate thresholds	bit-rate-high bit-rate-low
	Class	class	Packet rate thresholds	packet-rate-high packet-rate-low
		Total hosts thresholds	total-hosts-high total-hosts-low	
		Duration	duration	
		Remove action delay	remove-action-delay	

18.6.2.1 Filters

These filters are used with network protection:

Host grouping The default and most common host grouping is the timed-host grouping. This means that for this detection rule, the group consists of a single timed-host, for which we keep state and time for malicious activity.

The second host grouping is an all-hosts grouping which groups all of the timed-host detections (for all hosts over a detection type matching the given filters) and keeps state and aggregate statistics on all of them. This is very useful if you wish to take mitigation action when the number of hosts detected for a specific detection breach a threshold, or their combined packet-rate crosses a threshold, or their combined bit-rate passes a percentage threshold of the total bandwidth of your link. In these situations, an email-alert action is typically taken to notify a network administrator of abnormally high malicious activity.

Class	The class filter is a network class from the subnets.txt file and specifies which network or policy class is examined for analysis.
Transport	The transport is the transport protocol. Valid transport protocols are TCP, UDP. and ICMP.
Destination port	The destination port specifies which IP port is examined for analysis. Only packets from the specified port (TCP/UDP) or type (ICMP) will be considered.

18.6.2.2 Threshold Parameters

Any wmd-rule can have thresholds associated with it.

The purpose of a threshold is to control the circumstances under which mitigation actions are applied. While the flexibility exists to use any threshold in any rule, typically there are some thresholds that aren't applicable and some that are more applicable. For example, in a user-bandwidth wmd-rule, you would likely specify the bitrate threshold before you would specify a packet rate or total-recipients threshold. Likewise, in a syn-flood wmd-rule you would likely specify a packet rate threshold before specifying a bitrate threshold. Also, some thresholds, such as total hosts, and percent bitrate, are designed specifically with group all-hosts rules in mind.

High thresholds	High threshold parameters, when specified in a rule, instruct the aggregator to apply the mitigating actions associated with the rule immediately when the specified statistic value in the last event received is greater than the value specified by the threshold. For timed-host rules this comparison operation will be delayed until after the sample period timeout has expired and the aggregator has deemed the monitored host to be attacking.
Low thresholds	Low threshold parameters, when specified in a rule, instruct the aggregator to remove any applied mitigating actions associated with the rule immediately when the specified statistic value in the last event received is less than or equal to the value specified by the threshold. Note the statistic type specified by the low threshold must be the same type as that of the high threshold that was crossed prior to this event. This prevents a low packet rate from turning off the actions which were applied as the result of the bit-rate high threshold being crossed. Note that the removal of applied actions may be delayed by using the remove-action-delay timer.
Bitrate threshold (bit-rate-high, bit-rate-low)	A rule containing a bitrate threshold will apply the mitigation action(s) only if the bitrate of the host (over at least one detection period) exceeds the high threshold. The action will remain in place until the host drops its rate below the low threshold.
Packet rate threshold (packet-rate-high, packet-rate-low)	A rule containing a packet rate threshold will apply the mitigation action(s) if the packet rate of the host (over at least one detection period) exceeds the high threshold. The action will remain in place until the host drops its rate below the low threshold.
Total recipients threshold (total-recipients)	A rule containing a total recipients threshold will apply the mitigation action(s) if the total number of recipients that the host has sent email to over the duration of the detection session exceeds the specified total-recipients threshold.
Total hosts threshold (total-hosts-high, total-hosts-low)	The total hosts threshold is typically used in "group all-hosts" wmd-rules. It applies the rule's specified action(s) if the total number of actively malicious hosts is greater than the total-hosts-high threshold. The action(s) remain in place until the total number of actively malicious hosts drops below the total-hosts-low threshold.
Percent bitrate threshold (percent-bit-rate-high, percent-bit-rate-low)	The percent bitrate threshold is typically used in "group all-hosts" wmd-rules. It applies the rule's specified action(s) when the bitrate of the malicious traffic as compared to the bitrate of all traffic on the network exceeds the percent-bit-rate-high threshold as illustrated by this equation: `if M / T > P% then mitigate` Where: • M is the total malicious traffic (as a bitrate) • T is the total traffic (malicious and non-malicious, as a bitrate) • P is the wmd-rule's specified threshold.

18.6.2.3 Timer Parameters

Timer parameters are used to introduce a delay before applying or removing mitigating actions.

Duration timer When configured, this timer instructs the aggregator to wait the specified amount of time before applying mitigating actions. It may be specified on its own, without any high or low thresholds, or in combination with a set of high low thresholds. The duration timer is started when the detection session is created.

- Duration timer on its own: In this case the mitigating action will be applied immediately on the expiry of the timer. For timed-host rules this comparison operation may be disabled until after the sample period timeout has expired and the aggregator has deemed the monitored host to be attacking.
- Duration timer with one or more high/low thresholds: In this situation the behavior of the compare logic is modified such that the duration timer has to expire before the high threshold will be checked. This is equivalent to the logical end of the duration timer expiry and the crossing of a high threshold.

Remove action delay timer (remove-action-delay) When configured, the remove-action-delay timer instructs the aggregator to delay removing the applied mitigating actions until this period of time has passed. If during this time an event is received which results in the request that the mitigating actions be re-applied, the remove action delay timer will be stopped and the applied actions will remain in place undisturbed. If the detection sample timer expires while the remove action delay timer is running, and the aggregator did not receive any events for the host in that period, the aggregator will stop the remove action delay timer, remove all of the applied mitigating actions, and subsequently terminate the detection session.

18.6.2.4 Multiple Thresholds in a Rule

A wmd-rule may specify more than one set of high, low thresholds in a given rule.

In this situation the results of each of the individual comparisons are combined into the final value:

- The results of comparing with packet-rate-high, bit-rate-high, percent-bit-rate-high, and total-hosts-high will be logically or'd together in all cases.
- The results of comparing with packet-rate-low, bit-rate- low, percent-bit-rate- low, and total-hosts- low will be logically or'd together in all cases.
- The result of the duration threshold compare, for rules which specified duration with any high threshold, will be logically and'd with the result of the logical or of the high threshold compares.
- The result of the duration threshold compare, for rules which specified duration with any low threshold, will be ignored as will the result of the logical or of the low threshold compares. This means the low thresholds will be ignored for rules which include the duration threshold.

18.6.2.5 Escalating Thresholds

The policy file may specify many aggregator rules for the same detection type.

This leads to the situation where multiple rules may apply to a given detection session. Which rules to apply to a detection session is governed by the filters specified in each rule. Using the first event received for a new host for which the aggregator does not have any state, the aggregator will compare the detection type, host group, class (source or destination), transport type, destination port, and signature pattern against the filter values specified in each configured rule. All specified filters in a given rule must generate a match for the rule to be accepted by the detection.

To have an escalating threshold configuration, the set of rules are configured for a single detection type, which only differ in the specified high thresholds. For example:

```
wmd-rule 1 {address-scan group timed-host class \
 all-internal packet-rate-high 200 packet-rate-low 50} \
 {limit-packets src-ip ?src-ip} \
 {email-alert admin@myisp.com}
wmd-rule 2 {address-scan group timed-host class \
```

```
 all-internal packet-rate-high 150 packet-rate-low 50}\
 {email alert admin@myisp.com }
```

This policy has these effects:

- When the host is address scanning at a rate less than 150 packets per second (pps) the aggregator does not take any action.

  When the packet rate exceeds 150 pps, and the aggregator has deemed the host to be attacking based on the timed host configuration, the PTS triggers an email alert.

  If the host continues scanning and the packet rate increases such that the 200 pps threshold is crossed, the limit-packets action are applied to block the host's outgoing packets and the PTS triggers a second email alert.

  The actions applied against the host are removed once the packet rate drops below 50 pps, and the detection session is destroyed when the aggregator does not see any events for one complete sample period timeout.

From the operator's perspective, as the host being monitored escalates its malicious activity, the aggregator responds with escalating actions against that host.

18.6.3 Precedence for wdm-rule

When a number of rules all match a given set of detection parameters the most specific rule is selected to implement its mitigation action (if any).

This example illustrates two rules:

```
wmd-rule 1 {address-scan class all transport all} \
 {limit-packets src-ip ?malicious-host \
 dst-port ?malicious-port}
wmd-rule 2 {address-scan class all transport tcp dst-port \
 135} {limit-packets src-ip ?malicious-host}
```

In this example, address-scans on port 135 will match the second rule even though an address-scan on port 135 matches both rules' filters. This is because the second rule is more specific. Hosts address scanning on ports other than 135 will have their traffic blocked on only the port on which they are scanning. However, hosts address scanning on port 135 will have all of their traffic blocked.

This becomes more difficult when two rules are more specific than one another in different ways.

```
wmd-rule 1 {address-scan class all dst-port 135}
wmd-rule 2 {address-scan class all transport tcp}
```

In this example, a tcp address-scan on port 135 matches both rules, but in this case, the second rule matches. This is due to the order in which filters and parameters are evaluated when determining rule precedence.

The order in which filters and parameters are evaluated is:

1. Class
2. Transport
3. Source Port (src-port)
4. Destination Port (dst-port)
5. Source IP (src-ip)
6. Destination IP (dst-ip)
7. Pattern
8. PacketRateHigh (packet-rate-high)
9. PacketRateLow (packet-rate-low)
10. BitRateHigh (bit-rate-high)
11. BitRateLow (bit-rate-low)

12. PercentBitRateHigh (percent-bit-rate-high)

13. PercentBitRateLow (percent-bit-rate-low)

14. TotalHostsHigh (total-hosts-high)

15. TotalHostsLow (total-hosts-low)

16. TotalRecipientsHigh (total-recipients-high)

17. TotalRecipientsLow (total-recipients-low)

18. Duration

19. Remove Action Delay (remove-action-delay)

For high/low filters, since if specified differently they can both apply (unlike class, transport, port, and IP filters) it's important to know that if specified in two different rules, the higher value filter will take precedence over the lower value filter. For example, this rule:

```
wmd-rule 1 {address-scan class all-internal packet-rate-high 50pps packet-rate-low 40pps}
```

takes precedence over this one:

```
wmd-rule 2 {address-scan class all-internal packet-rate-high 45pps packet-rate-low 42pps}
```

18.6.4 Actions

A rule can contain one or more actions. Actions can contribute to detection or mitigation.

They are classified as either:

- Unconditional: detection and mitigation are not related. For example, a detection action can stand alone and mitigate all hosts on a specific port. All detection actions are unconditional.
- Conditional: information is passed from detection to mitigation. For example, detection can determine which port or user is problematic and pass this information to a mitigation action.

Note that:

- An action clause must have a action name as the first keyword in the clause.
- An action clause may have zero or more optional parameters following the action name keyword.

Actions are not specific to detections. In some cases the action may have more parameters than makes sense for the rule's detection. This is illustrated in the following example.

```
wmd-rule 1 {user-bandwidth} {limit-packets dst-ip ?dst-ip \
 dst-port ?dst-port}
```

In this example, ?dst-port will be replaced with port zero which will only block packets on port 0. This is probably not the intended effect.

The classification categories related to action performance are:

- High performance actions: have no impact or limit impact on element throughput. These actions can be used for mitigation in approximately 95% of all use cases.
- Low performance actions: have a high propensity to degrade processing power such that one hundred simultaneous mitigation instances will drop throughput by approximately 40%. Actions which are classified as low performance should only be used in specific circumstances and not generically deployed.

18.6.5 Conditional Action Parameters

These parameters can be used with conditional actions.

The “?” character indicates that information for the parameter is obtained from the detection action.

blocking-direction The direction with respect to the malicious host. Default is determined by the detection type. For example, for an address scan it is from-host, for a SYN flood attack it is the to-host.

Allowed values are: from-host|to-host.

dst-ip The destination IP to mitigate.

Allowed values are: ?dst-ip|?malicious-host|or a valid IP address.

dst-port The destination port to mitigate.

Allowed values are: ?dst-port|?malicious-port|or a valid port.

src-ip The source IP to mitigate.

Allowed values are: ?src-ip|?malicious-host|or any valid IP address.

src-port The source port to mitigate.

Allowed values are: ?src-port|?malicious-port|or any valid port.

transport The transport protocol.

Allowed values are: udp|tcp|icmp.

Conditional action values are:

from-host|to-host Conditional action values are used with blocking-direction parameters. They specify which host to block in a limit-host-packets rule. This is useful if you know if the detection part of the rule will always detect the initiator or responder of a flow.

?dst-ip The IP destination from the detected packets.

?src-ip The IP source address from the detected packets.

?dst-port The destination port from the detected packets.

?src-port The source port from the detected packets.

?malicious-host In rules where it is not consistently a source or destination, use these parameters to allow the detection to specify which host to mitigate. This is necessary for signature match since the detection engine may match a signature from source or destination. In other detections, malicious-hosts consistently returns either the source or destination. address-scan:

- source flow-flood: source
- syn-flood: destination
- user-bandwidth: destination
- spam: source

?malicious-port Typically the dst-port.

udp, tcp, icmp Supported transports.

This rule blocks all packets from the source address of any host found to be scanning port on 445.

```
wmd-rule 1 {address-scan class all-internal \
 transport tcp port 445} \
 {limit-packets src-ip ?src-ip}
```

This rule limits the traffic destined for any user that is the target of a DoS attack.

```
wmd-rule 2 {user-bandwidth class all-internal \
 threshold 5000000} \
 {limit-packets dst-ip ?dst-ip}
```

18.6.6 Action Event

The action-event mitigating action notifies an external application, such as the PCMM application manager, that is waiting for an event.

They syntax is:

```
wmd-rule <rule-number> {action_type}
{action-event event-name name parameters \
key/value key/value …}
```

18.6.7 Allow Packets

The allow-packets action ensures that all packets that match the specified parameters are explicitly allowed.

Classification: Low Performance

The allow-packets action defaults to a priority of 1 (highest priority). This action can be used to:

- Ensure that specific hosts are never affected by any other mitigation actions.
- Ensure that traffic on specific ports is never affected by any other mitigation action.
- Allow specific exclusions to other conditional actions for the same host.
- As a conditional action so that specific hosts can be mitigated.

The syntax is:

```
wmd-rule <rule-number> {condition} {allow-packets {order <n>}
 {dst-ip <n>} {dst-port <n>}
 {src-ip <n>} {src-port <n>} {transport tcp|udp|icmp|all}
 {tcp-flags <n>} {max-actions <n>}}
```

The parameters are:

transport	Specifies the transport protocol. Can be one of: tcp|udp|icmp|all. Default is all.
dst-ip	Destination IP address.
	Allowable values are: ip|?dst-ip|?malicious-host.
dst-port	Destination port.
	Allowable values are: tcp|udp.
max-actions	Maximum number of conditional actions that will be implemented by network protection.
	The default is: 1000.
order	Determines the order in which actions are applied.
	Allowable values are in the range: 1 - 10.
src-ip	Source IP address.
	Allowable values are: ip|?src-ip|?malicious-host

src-port	Source port. Allowable values are: tcp|udp.
tcp-flags	TCP flags. Allowable values are: fin|syn|rst|psh|ack|urg. The character ! denotes the absence of a flag

The allow-packets rule does not affect low performance mitigation actions. For example, if these two rules are implemented, rule 2 takes precedence and the IP address 192.168.1.1 is limited.

```
wmd-rule (allow 192.168.1.1}
wmd-rule 2 {limit-new-flows 192.168.1.1}
```

This rule allows any packets to or from host 10.0.0.1. The packets for this host are not mitigated.

```
wmd-rule 1 {allow-packets src-ip 10.0.0.1} {allow-packets dst-ip 10.0.0.1}
```

This rule ensures that HTTP traffic is never affected by mitigation.

```
wmd-rule 2 {allow-packets dst-port 80} {allow-packets src-port 80}
```

This rule blocks all packets from a host found to be address scanning except for packets destined to port 80.

```
wmd-rule 3 {address-scan group timed-host class all} \
    {allow-packets src-ip ?src-ip dst-port 80} \
    {limit-packets src-ip ?src-ip}
```

18.6.8 Limit Flows In

The limit-flows-in action limits the total number of outstanding flows that are destined to a host.

Classification: High Performance

The src-ip, src-port, dst-ip, and dst-port can be used as filters to apply the rule only to matching flows. This action instructs network protection to maintain state for each flow and is highly effective at mitigating DoS attacks. Connected flows are not impacted by the rule.

The reset-flows parameter is only intended to work with mitigating spam. Attempting to mitigate an address-scan, flow-flood, user-bandwidth, syn-flood or signature match using TCP RST is not only ineffective, but dangerous given the high number of packets transmitted by these attacks which will solicit a high amount of packets transmitted by the PTS.

Typically, the only malicious traffic for which sending TCP RSTs is appropriate is spam. No other application is recommended.

The syntax is:

```
wmd-rule <rule-number> {condition} {limit-flows-in <nflows>
 {order <n>} {dst-ip <n>} {dst-port <n>} {src-ip <n>} {src-port <n>}
 {reset-flows on|off}}
```

The parameters are:

dst-ip	Destination IP address. Allowable values are: ip|?dst-ip|?malicious-host.
dst-port	Destination port. Allowable values are: tcp|udp.
nflows	Number of flows permitted.
order	Determines the order in which actions are applied. Allowable values are in the range: 1 - 10.

reset-flows	Set to on to sends a TCP reset to each packet that requires mitigation. Default is off.
src-ip	Source IP address.
	Allowable values are: ip|?src-ip|?malicious-host
src-port	Source port.
	Allowable values are: tcp|udp.

This rule limits every host to two incoming HTTP connections at a time:

```
wmd-rule 1 {limit-flows-in 2 dst-port 80}
```

18.6.9 Limit Flows Out

The limit-flows-out action limits the total number of outstanding flows that originate from a host.

Classification: High Performance

The src-ip, src-port, dst-ip, and dst-port can be used as filters to apply the rule only to matching flows. This action instructs network protection to maintain state for each flow and is highly effective at mitigating scanning worms without impacting non-infected hosts. Connected flows are not impacted by the rule.

The reset-flows parameter is only intended to work with mitigating spam. Attempting to mitigate an address-scan, flow-flood, user-bandwidth, syn-flood or signature match using TCP RST is not only ineffective, but dangerous given the high number of packets transmitted by these attacks which will solicit a high amount of packets transmitted by the PTS.

Typically, the only malicious traffic for which sending TCP RSTs is appropriate is spam. No other application is recommended.

The syntax is:

```
wmd-rule <rule-number> {condition} {limit-flows-out <nflows>
 {order <n>} {dst-ip <n>} {dst-port <n>} {src-ip <n>} {src-port <n>}
 {reset-flows on|off}}
```

The parameters are:

dst-ip	Destination IP address.
	Allowable values are: ip|?dst-ip|?malicious-host.
dst-port	Destination port.
	Allowable values are: tcp|udp.
nflows	Number of flows permitted.
order	Determines the order in which actions are applied.
	Allowable values are in the range: 1 - 10.
reset-flows	Set to on to sends a TCP reset to each packet that requires mitigation. Default is off.
src-ip	Source IP address.
	Allowable values are: ip|?src-ip|?malicious-host
src-port	Source port.
	Allowable values are: tcp|udp.

This rule limits every host to 10 outgoing HTTP connections at a time.

```
wmd-rule 1 {limit-flows-out 10 dst-port 80}
```

This rule blocks and sends a TCP RST packet in response to each packet which belongs to the 6th or greater flow of src-ip 1.1.1.1.

```
wmd-rule 2 {limit-flows-out 5 src-ip 1.1.1.1 transport tcp reset-flows on}
```

18.6.10 Limit Host Packets

The limit-hosts-packets action limits packets going to a particular host or coming from a particular host.

For more precise packet limiting, transport and or destination port can be used. This action allows policies to be written in a general way, but have a very specific action depending on the detection.

Classification: High Performance if the rate parameter is set to 0, otherwise, Low Performance

The syntax is:

```
wmd-rule <rule-number> {condition} {limit-host-packets
 {blocking-direction ?blocking-direction|to-host|from-host} {ip <n>}
 {malicious-port ?malicious-port|?src-port|?dst-port|<n>|all}
 {transport ?transport|udp|tcp|icmp|all} {rate <n>}}
```

The parameters are:

transport	Specifies the transport protocol. Can be one of: tcp|udp|icmp|all. Default is all.
blocking-direction	The direction with respect to the malicious host. Default is determined by the detection type. For example, for an address scan, it is from-host, for a SYN flood attack it is the to-host. Can be one of: ?blocking-direction|to-host|from-host
ip	The IP address of the host of interest. Can be one of: ?src-ip|?dst-ip|ip|?malicious-host
malicious-port	The port of malicious intent. This is usually the destination-port, but could be the source-port. Can be one of: ?malicious-port|?src-port|?dst-port|port|all

This rule blocks packets based on an address scan detection. It blocks all packets that match the transport and destination port from the host detected doing the address scan.

```
wmd-rule 1 {address-scan class all} {limit-host-packets}
```

This rule blocks packets based on signature detections. It blocks all packets that match the transport and destination port coming from the infected host or going to the targeted host depending on the signature detected: worm or trojan, respectively.

```
wmd-rule 2 {signature-match class all pattern all} \
 {limit-host-packets}
```

This rule blocks packets based on spam detection. It blocks all packets regardless of destination port or transport coming from the host detected as spending spam.

```
wmd-rule 3 {spam class all-internal} \
 {limit-host-packets dst-port all transport all}
```

Limit-host-packets can also be used as an unconditional rule. As such, blocking-direction and IP must be specified and transport and dst-port set to all.

```
wmd-rule 4 {limit-host-packets ip 1.0.0.2 \
 blocking-direction to-host}
```

18.6.11 Limit New Flows

The limit-new-flows action limits outgoing flows from each host.

Classification: Low Performance

This action applies the limit-packets action to TCP SYN packets. It is generally used to mitigate DoS flows floods or DoS SYN floods and can effectively mitigate malicious traffic without impacting regular traffic. The limit-new-flows action can be used as a conditional action if the source or destination address/port is passed in from a detection action in the same rule. Using this action to block or shape infected hosts is relatively expensive since a new action is created for each host. The limit-flows-out and limit-flows-in rule offers significantly better performance when the number of hosts is large.

For more information about conditional actions, see *Conditional Action Parameters* on page 228.

The syntax is:

```
wmd-rule <rule-number> {condition} {limit-new-flows {order <n>}
 {rate <n>} {dst-ip <n>}   {dst-port <n>}
 {src-ip <n>} {src-port <n>}}
```

The parameters are:

dst-ip	Destination IP address. Allowable values are: ip|?dst-ip|?malicious-host.
dst-port	Destination port. Allowable values are: tcp|udp.
order	Determines the order in which actions are applied. Allowable values are in the range: 1 - 10.
rate	Optional. Rate in packets/second or bits/second depending upon the context. If rate is used for a mitigation action, it will default to 0. A rate of 0 indicates that all packets will be blocked. Allowable values are: bps|kbps|mbps|pps|kpps|mpps|ip|?src-ip|?malicious-host
src-ip	Source IP address. Allowable values are: ip|?src-ip|?malicious-host
src-port	Source port. Allowable values are: tcp|udp.

This example limits TCP SYN packets that from host 10.0.0.1 to 20.0.0.1 on port 80 to 2 packets per second.

```
wmd-rule 1 {limit-new-flows rate 2 src-ip 10.0.0.1 \
 dst-ip 20.0.0.1 dst-port   80}
```

This action blocks all TCP SYN packets from the host that was detected by the detection action in a rule. The host was the destination in the detection action. The maximum number of conditional actions allowed and the timeout are set to their defaults.

```
wmd-rule 2 {limit-new-flows src-ip ?dst-ip}
```

18.6.12 Limit-packets

The limit-packets action blocks or shapes all packets that match the specified parameters.

Classification: High Performance if the rate parameter is set to 0, otherwise, Low Performance

This action is generally used to block packets involved in a DoS attack. It is relatively inexpensive to implement. The limit-packets action can used as a conditional action if the source or destination address/port is passed in from a detection action in the same rule.

The reset-flows parameter is only intended to work with mitigating spam. Attempting to mitigate an address-scan, flow-flood, user-bandwidth, syn-flood or signature match using TCP RST is not only ineffective, but dangerous given the high number of packets transmitted by these attacks which will solicit a high amount of packets transmitted by the PTS.

Typically, the only malicious traffic for which sending TCP RSTs is appropriate is spam. No other application is recommended.

For more information about conditional actions, see *Conditional Action Parameters* on page 228.

The syntax is:

```
wmd-rule <rule-number>{condition} {limit-packets {order <n>}
 {rate <n>} {dst-ip <n>}
 {dst-port <n>} {src-ip <n>} {src-port <n>}
 {transport tcp|udp|icmp}
 {tcp-flags <n>} {max-actions <n>} {reset-flows on|off}}
```

The parameters are:

transport	Specifies the transport protocol. Can be one of: tcp|udp|icmp|all. Default is all.
dst-ip	Destination IP address.
	Allowable values are: ip|?dst-ip|?malicious-host.
dst-port	Destination port.
	Allowable values are: tcp|udp.
max-action	Maximum number of conditional actions that will be implemented by network protection. Applies only to conditional actions
order	Determines the order in which actions are applied.
	Allowable values are in the range: 1 - 10.
reset-flows	Set to on to sends a TCP reset to each packet that requires mitigation. Default is off.
src-ip	Source IP address.
	Allowable values are: ip|?src-ip|?malicious-host
src-port	Source port.
	Allowable values are: tcp|udp.
tcp-flags	TCP flags.
	Allowable values are: fin|syn|rst|psh|ack|urg. The character ! denotes the absence of a flag

This rule limits packets from host 10.0.0.1 to 20.0.0.1 on port 80 to 150 packets per second

```
wmd-rule 1 {limit-packets rate 150 src-ip 10.0.0.1 dst-ip \
20.0.0.1 dst-port 80}
```

This rule blocks all TCP packets from a host that was detected by the detection action in a rule. It limits the maximum number of conditional actions to 5 and time the rule out after 30 seconds of inactivity.

```
wmd-rule 2 {limit-packets src-ip ?src-ip \
transport tcp max-actions 5 timeout 30}
```

This rule blocks and sends a TCP RST for TCP flows for a server and dst-port for which address-scanning has been detected.

```
wmd-rule 2 {address-scan transport tcp} {limit-packets \
src-ip ?src-ip dst-port ?dst-port transport tcp \ reset-flows on}
```

The rule blocks and sends a TCP RST for each packet for which 1.1.1.1 is the client.

```
wmd-rule 3 {limit-packets src-ip 1.1.1.1 reset-flows on}
```

18.6.13 Reroute

The reroute action instructs the CND to send a BGP update to the appropriate routers for the purpose of rerouting malicious traffic to the CND for mitigation.

Classification: High Performance

This is typically used in a situation where the CND is not in-line (in a selective in-line setup). This action selects the given IP/subnet to be in-line.

This action can be used as a conditional action to capture traffic only after the detection action has discovered a problem. It can also be used as an unconditional action to permanently in-line traffic destined to the given IP/subnet.

Direction can be one of to-internal, to-external, or auto. Typically auto is used to indicate to the CND that it can determine which router should be BGP updated; however, this can be overridden by specifying to-internal or to-external.

Sandvine recommends that you do not specify the direction parameters unless you are certain it is what you want. Use auto, the default value for redirection and to indicate that you want the network class manager to determine which router should be updated to allow the reroute to take place.

The syntax is:

```
wmd-rule <rule-number> {condition} {reroute dst-ip <ip>
    {direction auto|to-internal|to-external}}
```

The parameters are:

dst-ip	Destination IP address. Allowable values are: ip|?dst-ip|?malicious-host.
direction	Specifies whether the dst-ip is internal or external. Allowable values are: to-internal|to-external|auto. The default is auto.

This action reroutes all traffic destined for 10.1.1.0/24 to the CND.

```
wmd-rule 1 {reroute dst-ip 10.1.1.0/24}
```

This action reroutes all traffic destined for 10.1.1.1 to the CND by sending a BGP update message only to routers configured to handle messages that affect internal IPs. This allows the rule definition to override the network class determination of which router this BGP Update message should be sent to.

```
 wmd-rule 2 {reroute dst-ip 10.1.1.1 direction to-internal}
```

This action reroutes all traffic to a host found by a detection action.

```
wmd-rule 3 {reroute dst-ip ?dst-ip}
```

18.6.14 Nullroute

The nullroute action instructs the CND to send a BGP update message to the appropriate routers for the purpose of nullrouting (or blackholing) malicious traffic.

Classification: High Performance

This is a more heavy-handed approach to mitigation than the reroute action. While the reroute action reroutes the traffic for mitigation, the nullroute action makes the traffic disappear completely.

The syntax is:

```
wmd-rule <rule-number> {condition} {nullroute dst-ip ip {direction auto|to-internal|to-external}}
```

The parameters are:

dst-ip	Destination IP address. Allowable values are: ip|?dst-ip|?malicious-host.
direction	Specifies whether the dst-ip is internal or external. Allowable values are: to-internal|to-external|auto. The default is auto.

This action nullroutes all traffic destined for 10.1.1.0/24.

```
wmd-rule 1 {nullroute dst-ip 10.1.1.0/24}
```

This action nullroutes all traffic destined for 10.1.1.1 to the CND by sending a BGP Update message only to routers configured to handle messages that affect internal IPs. This allows the rule definition to override the network class determination of which router this BGP update message should be sent to.

```
wmd-rule 2 {nullroute dst-ip 10.1.1.1 direction to-internal}
```

This action nullroutes all traffic to a host found by a detection action.

```
wmd-rule 3 {nullroute dst-ip ?dst-ip}
```

18.6.15 Subscriber Attribute, Including Captive Portal

Subscriber attribute is used to mark a subscriber with a specific attribute.

Classification: High Performance

The traffic for that subscriber can then be manipulated by the PTS. The primary use of the subscriber attribute is to mark the subscriber as being 'infected' so that their HTTP traffic is redirected to the captive portal.

The network protection subscriber-attribute action instructs the system to set a specific attribute for the specified subscriber.

The network protection subscriber-attribute action simply sets the specified attribute in the subscriber attribute database. The PTS policy defines how the attribute will affect the traffic for the subscriber.

In the event that a subscriber attribute set by CND has its expiry time externally modified, CND will not reset the expiry time.

The syntax is:

```
wmd-rule <rule-number> {condition} {subscriber-attribute
    host <n> attribute <x> value <y> default_value <n> {duration <n>}{persistent}}
```

The parameters are:

host	The IP address of the subscriber, or one of the "conditional action parameters" ?src-ip ?dst-ip or ?malicious-host.
attribute	The name of the attribute that should be set. The attribute name must correspond to a valid attribute name defined in the PTS policy file.
value	The value that should be set for the attribute when the action is to be applied. The value must match one of the allowed values for the attribute defined in the PTS policy file.
duration	This value only applies to the persistent parameter.

	Allowable values are: s|seconds m|minutes h|hours d|days. For example: 15s, 30minutes, 1h. Default is 30minutes.
persistent	If this parameter is included then network protection will not remove the attribute from the subscriber. This ensures that the attribute will be set for the duration time after the action is disabled.
default_value	The value that should be set for the attribute when the action is not being applied. The value must match one of the allowed values for the attribute defined in the policy file. Specifying this parameter has the side effect of including the persistent parameter.

This rule marks a subscriber as being infected whenever they are found to be address scanning. The subscriber attribute will be removed when the host stops address scanning.

```
wmd-rule 1 {address-scan group timed-host class \
    all-internal} \
    {subscriber-attribute host ?malicious-host \
    attribute infected value true}
```

This rule marks the subscriber with the IP address of 1.0.0.1 as being infected.

```
wmd-rule 2 {subscriber-attribute host 1.0.0.1 attribute infected value true}
```

The corresponding rule in the PTS policy file would instruct the PTS to captive portal the traffic for that host. The host's HTTP traffic would be redirected to the specified web site. See *Captive Portal Overview* on page 147

```
# Network protection captive portal
attribute 'infected' values 'true' 'false'
if protocol "http" and client "infected"="true" then \
    captive_portal "http://myportal.domain.com/infected.rvt"
```

18.6.16 Email Alert

Email alerts are intended to notify an ISP or Sandvine's Security Operations Services when a potentially critical detection occurs.

Classification: High Performance

Detection Types: Address scan, Flow flood, SYN flood, User bandwidth, Spam Signature detection

Detection Groups: timed-host, group-host

Email alerts are not intended to email a subscriber and should only be used for detections that may require immediate attention.

An email alert is sent when a specific set of conditions are met. By default, the email displays "WDTM Alert" in the subject and the email body contains the rule from the policy file that enabled the alert and a report from Network Demographics. Currently there are six detection types and two detection groupings, as listed below. A different report is generated for each detection type and grouping with the exception of flow floods. Flow flood timed-host reports can be either source IP-address or destination IP-address based.

Note:
The default configuration assumes that network protection is deployed on the same element as the Network Demographic Reports server. If this is not the case, and the reports server is on another element, then run the `set config nds security trusted-ip-address` CLI configuration command on the element where the reports server resides. This variable takes a comma-delimited list of all remote WDTM PTS IP addresses that generate alert emails. For more information, refer to the chapter entitled “Network demographic server configuration” in the current version of the *SPB Administration Guide*.

For a timed-host detection group, there is an optional parameter ip-address. If ip-address is specified the report contained in the email will be for the specified subscriber.

For a multiple host detection group, there are two optional parameters, hosts-to-display and report-period. hosts-to-display is the number of hosts to appear in the report contained in the email. The worst <n> hosts will be in the report. report-period specifies the amount of time the report will go back for.

It is not possible to specify both ip-address and hosts-to-display.

The syntax is:

```
wmd-rule <rule-number> {{condition} email-alert to-address
 {cc-address "<string>"}
 {bcc-address "<string>"} {from-address "<string>"}
 {subject "<string>"} {email-message "<string>"}
 {report-period s|seconds|m|minutes|h|hours|d|days}
 {hosts-to-display <n>} {ip-address ip-addr}
 {report-detection-type address-scan-single|
  address-scan-group|
  flow-flood-single-src|
  flow-flood-single-dst|
  flow-flood-group|
  user-bandwidth-single|
  signature-single|
  spam-single|
  syn-flood-single|
  syn-flood-group}}
```

The parameters are:

bcc-address Email a blind carbon copy of the alert to the specified email address(es).

Allowed values are standard email addresses. If there are multiple email addresses, separate them with a comma.

email-message Email a carbon copy of the alert to the specified email address(es).

Allowed values are standard email addresses. If there are multiple email addresses, separate them with a comma.

email-message Text to appear before the report in the email body.

Allowed values are any alphanumeric character or set of characters. Special characters like line-feeds or quotes will cause the policy to fail to load. The default is the rule from the policy file that triggered the email alert.

from-address Sender's email address.

Allowed values are standard email address.

hosts-to-display For a multiple host detection group, the number of hosts to display in the report.

Allowed values are integers greater than 0.

ip-address For a timed-host detection group, the specific subscriber on which to report.

Allowed values are a valid IP address. When used as a conditional action, ?src-ip, ?dst-ip, and ?malicious-host can be used.

report-detection-type Type of report.

If the rule is unconditional, one of these must be specified:

- address-scan-single
- address-scan-group
- flow-flood-single-src (report on source of flow flood)
- flow-flood-single-dst (report on destination of flow flood)
- flow-flood-group
- signature-single
- spam-single
- syn-flood-group

	If the rule is conditional, the detection type indicated by rule is used.
report-period	This applies to a multiple host detection group only. This is the time range preceding the detection event on which to report. For example if the report-period is 15 minutes, the report will show the activity for the 15 minutes preceding the event. Allowed values are period specifiers. Default is 15 minutes.
subject	Subject that appears in the email subject field. Allowed values are any alphanumeric character or set of characters. Special characters like line-feeds or quotes will cause the policy to fail to load. The default is the rule from the policy file that triggered the email alert. Default is WDTM alert.
to-address	Recipient email address. Required parameter that must be specified. Allowed values are standard email addresses. If there are multiple email addresses, separate them with a comma.

Example:
In this example, when an email alert action is taken, a report is generated and emailed to securityoperations@sandvine.com. Default values are used for all optional parameters. The detection type and group, and report to send are determined by the plug-in.

```
wmd-rule 1 {email-alert to-address securityoperations@sandvine.com}
```

When an email alert action is taken, a report will be generated and emailed to securityoperations@sandvine.com. Default values are used for all optional parameters except for ip-address. The detection type is determined by the plug-in. Detection group is timed-host. The report still depends on the detection type, but only contains information for the subscriber with IP 1.1.1.1.

```
wmd-rule 2 {email-alert to-address securityoperations@sandvine.com ip-address 1.1.1.1}
```

When an email alert action is taken, a report will be generated and emailed to securityoperations@sandvine.com. Default values are used for all optional parameters except for hosts-to-display and report-period. The detection type is determined by the plug-in. Detection group is multiple hosts. The report depends on the detection type, but will contain specifics for the hosts-to-display worst offenders and a summary of all host activity from the time the alert is sent to report-period.

```
wmd-rule 3 {email-alert to-address \
    securityoperations@sandvine.com \
    hosts-to-display 10 report-period 10minutes}
```

In this example a report will be sent immediately to securityoperations@sandvine.com and will contain information only for the subscriber at the indicated IP address for the indicated report period.

```
wmd-rule 4 {email-alert to-address \
    securityoperations@sandvine.com \
    ip-address 1.1.1.1 report-period 10minutes \
    report-detection-type span-single}
```

18.6.17 Action-event

The action-event action is used to trigger an event.

This is typically used in conjunction with a PCMM action. For more information on action-event refer to the Sandvine PacketCableTM Multimedia Application Manager application note or contact Sandvine Customer Support.

19

Configuring the Network Protection Module

19.1 Network Protection Configuration

CLI configuration commands for network protection are:

CLI Command	Description	Default
set config network-protection minimum-rule-priority <int:0..>	Determines which rules will be run: rules having a priority equal to or less this value are enabled. For example, setting this value to 5 enables all rules with priorities 1, 2, 3, 4 and 5 but disables all rules with priorities 6, 7, 8, 9, and 10. You can decrease this value to prevent network protection from automatically implementing rules posted by Sandvine. Set this value to 0 to disable all network protection policies that are part of the policy.sandvine.conf file.	10
set config network-protection mitigation enable <false|true>	Allows mitigation to determine whether or not network protection will implement mitigation actions. If set to true then any rules entered by Sandvine or by the ISP (in other network protection policy files) will be implemented completely. If set to false, then any rules will only be implemented if they are detection rules. The mitigation component of the rules will be disabled.	true
set config network-protection max-actions <int:0..>	Defines the maximum number of mitigation actions to be implemented per module. This variable is set globally for all modules in the element. For example, if this variable is set to 100, this means that each module can implement up to 100 mitigation actions.	100

19.1.1 Spam detection configuration

The CLI configuration commands used to configure spam detection are:

CLI Command	Description	Default
set config network-protection max-smtp-hosts <int:0..>	Sets the maximum number of SMTP hosts spam detection will track and maintain state for. For deployments where only SMTP traffic is being managed (Mail Server Defender deployments), the maximum recommended value is 30000.	5000
set config network-protection max-ports-per-smtp-host <int:0..>	The maximum number of ports per SMTP host to be monitored for spam. Spam will be detected and mitigated on a per port	5

CLI Command	Description	Default
	basis. If this value is decreased and a SMTP host is already monitoring more ports than the new value, all monitored ports will continue to be monitored until SMTP is no longer being sent on those ports.	

19.1.2 Email Alerts Configuration Variables

To use email alerts you must first configure the report-server using this CLI command:

CLI Command	Description	Default
set config network-protection email-alert report-server <server>	The report server the PTS will trigger to send the email alerts	usually the NDS

These commands configure the maximum number of email alerts that can be burst out in a given time. The default is 10/600. This means the maximum number of email messages sent in any 600 second windows is 10.

CLI Command	Description	Default
set config network-protection email-alert rate-limit <int:0..>	The maximum number of emails that can be sent in rate-period seconds.	10
set config network-protection email-alert rate-period <int:1..>	The period, in seconds, in which a maximum of rate-limit emails can be sent.	600

19.1.3 Configuring Email Alert Username and Password

The element on which the Central Node Daemon (CND) is installed must be able to connect to the Network Demographics Reports Server to generate email alerts. The username and password set for the CND must match a username and password that has been enabled on the Network Demographics Reports Server.

1. Identify which user names and password have been enabled on the Network Demographics Reports Server.

   The user name is usually a default Sandvine user (sv_user, sv_service, or sv_admin).

2. On the central PTS run: these commands:

```
PTS> configure
PTS# set config network-protection email-alert username <username>
PTS# set config network-protection email-alert password <password>
```

3. After you have completed your configuration, commit the changes. The system automatically reloads, reboots or restarts, which may impact service. Run: `commit`

20

Configuring VoIP Measurements and Management

20.1 Configuring VoIP Usage

Statistics such as bandwidth, minutes, users, and providers can be tracked for VoIP traffic.

20.1.1 Identifying additional VoIP providers

You can use the CLI to add recognition for VoIP signatures.

Over a hundred providers have been configured and exist in the /usr/local/sandvine/etc/data/ folder. All other providers that are detected are bundled under “unknown” until specific signatures are added for them, either via an updated software release or by adding the signatures using the CLI.

1. To identify additional VoIP providers, run these commands:

   ```
   PTS> configure
   PTS# add config service protocol voip provider <name> uri <uri>
   ```

   where <name> is the name of the VoIP provider and <uri> is the provider’s URI. The URI can use a regular expression for character matching. In the example, ^ matches at the beginning of the line and $ matches at the end of the line.

   For example:

   ```
   PTS> configure
   PTS# add config service protocol voip provider Vonage uri vonage.net
   PTS# add config service protocol voip provider Free World Dialup uri ^pulver.fwd.net$
   ```

2. After you have completed your configuration, commit the changes. The system automatically reloads, reboots or restarts, which may impact service. Run: `commit`.

20.1.2 Identifying Unknown VoIP Providers

SIP and MGCP traffic that is not from any of the providers configured as detailed in the previous topics is logged as “unknown”.

An option is automatically enabled to log the URIs (IP address or DNS name) of the server with which a subscriber of an unknown provider is communicating.

Run the `show service protocol voip unknown-providers` CLI command on the PTS to retrieve the list of unknown VoIP providers.

Run the `clear service protocol voip unknown-providers` CLI command on the PTS to clear the list of unknown providers.

20.1.3 Enabling VoIP

Protocol and VoIP statistics are reported only if logging is enabled.

Logging is automatically enabled if an if true then count demographic rule is present in the policy.conf file. For information on creating a rule with a count action, refer to *SandScript Grammar* on page 24.

20.1.4 Blocking VoIP

Certain jurisdictions impose regulatory requirements that CSPs block unauthorized VoIP providers. A "dropped calls" field is present in the VoIP statistics. This field is reported as the number of calls that were dropped on the Blocked Calls by Provider and Blocked Calls by Protocol Network Demographic reports.

Blocking the SIP protocol blocks the SIP control flows as well as SIP data. When the control flows are blocked, no data flows are set up. To determine the number of calls which have been blocked, block only SIP data. This will allow the data flows to be set up and then subsequently blocked, which results in the number of calls blocked being logged to the database.

Note that VoIP statistics must be enabled for reporting to occur.

Example:
This example blocks all SIP calls and properly populates the dropped calls field.

```
if protocol "sip_data" then block
```

These examples are policy statements relevant to VoIP blocking.

This policy blocks SIP calls belonging to provider 'exampleProvider'.

```
if protocol "sip_data" and provider "exampleProvider" then block
```

This policy blocks SIP calls not belonging to provider "exampleProvider".

```
if protocol "sip_data" and not provider "exampleProvider" then block
```

The following policy will block and track blocking statistics for MGCP calls.

```
 if protocol "mgcp_data" then block
```

This policy blocks h323 calls.

```
 if protocol "h323_data" then block
```

21

Troubleshooting SandScript

21.1 Tips for Using Unique By

If your policy is over counting, over shaping or over limiting (doing too much of any of these actions), then it may be that one or more of your unique by clauses are not unique.

Using shapers, limiters or measurements that are unique by either a string field or by multiple expression fields may result in incorrect behavior. In such cases, different unique by instances can collide and result in incorrect values being returned by policy. The uniqueness of a unique by clause is guaranteed when the clause contains:

- a single integer expression.
- a single boolean expression.
- a subscriber expression and a classifier expression.
- a protocol expression and one or two classifier expressions.
- one to four classifier expressions.

If a unique by clause is used in policy that does not follow the above rules that guarantee uniqueness, then a warning will be raised when the policy is reloaded.

21.2 Troubleshooting the PTS

The following section provides additional troubleshooting information for a PTS element. If the problem cannot be resolved, contact Sandvine customer support for assistance.

21.2.1 Message Logs

Most error messages are sent to syslog.

Use the `show config system log remote-server` CLI command to determine the configured location for the sys log messages.

The default location of the logs is `/var/log/svlog`. For more information on log messages, refer to the *PTS CLI/Alarms Reference Guide*.

21.2.2 Unidentified Traffic

If shaping traffic, and the router that is sitting upstream from the PTS is not able to identify marked traffic it may be that the TOS fields are not set or are incompatible. Verify policy.conf is correct and execute reload. For more information, refer to *File Validation through Reload* on page 21.

21.2.3 Informative CLI Commands

Sandvine provides system information through CLI commands.

CLI command	Description
`show system overview`	Shows an overview of what the PTS or SDE is doing.
`show system processes`	Shows system processes running on the controller for all modules.
`show system resources`	Shows a list of system resources for the system, a specific resource ID, controller or module. System resources include hard disk space, memory, flows, and other resources that, if exhausted, will impact the proper functioning of the system.

A

Network Protection Use Cases

A.1 General Mitigation

These policy examples are of general mitigation types.

To block all port 445 traffic:

```
wmd-rule 1 {limit-packets dst-port 445}
```

To block all port 135 traffic:

```
wmd-rule 2 {limit-packets dst-port 135}
```

To block all port 139 traffic:

```
wmd-rule 3 {limit-packets dst-port 139}
```

To block all UDP traffic on port 1026:

```
 wmd-rule 4 {limit-packets transport udp dst-port 1026}
```

To allow X concurrent sessions of SMTP traffic from each subscriber to their mail server (x.x.x.x):

```
wmd-rule 6 {limit-flows-out X dst-ip x.x.x.x dst-port 25}
```

To force DNS traffic to specific servers (x.x.x.x and y.y.y.y). This requires N+1 rules where N is the number of servers.

```
wmd-rule 7 {allow-packets order 3 dst-port 53 dst-ip x.x.x.x}
wmd-rule 8 {allow-packets order 2 dst-port 53 dst-ip y.y.y.y}
wmd-rule 9 {limit-packets order 1 dst-port 53}
```

To exempt angel-ip specific hosts from detection. Note that this will also prevent the hosts from being protected from attack.
detection-config

```
10 {angel-ip address 1.1.1.1}
```

To exempt angel-ip specific hosts from mitigation. Note that this will only exempt the host from traffic mitigation. The host will not be exempt from non-traffic mitigation actions like captive portal and bgp nullroute.

```
wmd-rule 11 {allow-packets order 10 src-ip 1.1.1.1}
```

A.2 Address Scans

Much of the aggregator-config and detection-config rules specified below are already configured in policy.sandvine.conf. They are documented here so that you understand the full policy configuration and the interaction between policy rules.

Use this rule to block address scans on specific ports from any hosts (135, 139, 445, 1026 ...). The action is removed after 2 minutes of no address scans.

```
aggregator-config 12 {address-scan sample-period 1m \
 timed-host-percent 40 detection-timeout 2m}
detection-config 13 {address-scan class all period 15 \
 threshold 15}
wmd-rule 14 {address-scan class all dst-port 135} \
 {limit-host-packets}
wmd-rule 15 {address-scan class all dst-port 139} \
 {limit-host-packets}
wmd-rule 16 {address-scan class all dst-port 145} \
 {limit-host-packets}
wmd-rule 17 {address-scan class all dst-port 1026} \
 {limit-host-packets}
```

This rule blocks address scans on specific ports from any internal hosts.

```
aggregator-config 12 {address-scan sample-period 1m \
 timed-host-percent 40 detection-timeout 2m}
detection-config 18 {address-scan class all-internal period 15 \
 threshold 15}
wmd-rule 19 {address-scan class all-internal dst-port 135} \
 {limit-host-packets}
wmd-rule 20 {address-scan class all-internal dst-port 139} \
 {limit-host-packets}
wmd-rule 21 {address-scan class all-internal dst-port 145} \
 {limit-host-packets}
wmd-rule 22 {address-scan class all-internal dst-port 1026} \
 {limit-host-packets}
```

This rule captive portals and blocks non-port 80 and 53 traffic for hosts address scanning. This example assumes that captive portal is fully configured. Note that this rule will only captive portal internal subscribers. If you attempt to captive portal an external host, a warning messages is logged in /var/log/svlog.

Note:
The order of these rules is such that the allow rules do not override mitigation done by other rules (which will use the default order of 5).

```
aggregator-config 12 {address-scan sample-period 1m \
 timed-host-percent 40 detection-timeout 2m}
detection-config 18 {address-scan class all-internal period 15 \
 threshold 15}
wmd-rule 23 {address-scan class all-internal} \
 {allow-packets order 6 src-ip ?src-ip dst-port 53} \
 {allow-packets order 6 src-ip ?src-ip dst-port 80} \
 {limit-packets order 7 src-ip ?src-ip} \
 {subscriber-attribute host ?src-ip attribute infected value \
 true}
```

This rule blocks all traffic for address scanners, but always allows traffic to the captive portal server (x.x.x.x). This is more efficient than the previous example since this requires N+1 traffic mitigation rules per detection, whereas the previous example requires 3N rules.

```
aggregator-config 12 {address-scan sample-period 1m \
 timed-host-percent 40 detection-timeout 2m}
detection-config 18 {address-scan class all-internal period 15 \
 threshold 15}
wmd-rule 24 {allow-packets dst-ip x.x.x.x dst-port 80}
wmd-rule 25 {address-scan class all-internal} \
 {limit-packets src-ip ?src-ip} \
 {subscriber-attribute host ?src-ip attribute infected \
 value true}
```

This rule will block address scans on all ports, except port 80, from any internal host.

```
aggregator-config 12 {address-scan sample-period 1m \
 timed-host-percent 40 detection-timeout 2m}
detection-config 18 {address-scan class all-internal period 15 \
 threshold 15}
wmd-rule 26 {address-scan class all-internal} \
 {limit-host-packets}
wmd-rule 27 {address-scan class all-internal dst-port 80}
```

This rule emails Sandvine's Security Operations Services when any conditional action is applied. To achieve this, just append an email-alert action to all of the wmd-rules. This rule:

```
wmd-rule 40 {syn-flood class all-internal} {limit-host-packets}
```

Becomes:

```
wmd-rule 40 {syn-flood class all-internal} {limit-host-packets} \
 {email-alert to-address securityOperations@sandvine.com}
```

This rule emails Sandvine's Security Operations Services when total address scan bandwidth exceeds 1% of the total traffic.

```
wmd-rule 41 {address-scan group all-hosts \
 percent-bit-rate-high 1 percent-bit-rate-low .75} \
 {email-alert to-address securityOperations@sandvine.com}
```

This rule emails Sandvine's Security Operations Services when total hosts scanning from a single network class on any port# (not specified) exceeds 100 hosts. To do this, a rule is required for each network class/port combination.

```
wmd-rule 42 {address-scan group all-hosts class A dst-port 135 \
 total-hosts-high 100 total-hosts-low 75} \
 {email-alert to-address securityOperations@sandvine.com}
wmd-rule 43 {address-scan group all-hosts class A dst-port 445 \
 total-hosts-high 100 total-hosts-low 75} \
 {email-alert to-address securityOperations@sandvine.com}
wmd-rule 44 {address-scan group all-hosts class B dst-port 135 \
 total-hosts-high 100 total-hosts-low 75} \
 {email-alert to-address securityOperations@sandvine.com}
wmd-rule 45 {address-scan group all-hosts class B dst-port 445 \
 total-hosts-high 100 total-hosts-low 75} \
 {email-alert to-address securityOperations@sandvine.com}
# ... continue for each class/port combination
```

This rule emails Security Operations Services when total hosts scanning from a single network class on any port exceeds 100.

```
wmd-rule 46 {address-scan group all-hosts class A \
 total-hosts-high 100 total-hosts-low 75} \
 {email-alert to-address securityOperations@sandvine.com}
```

This rule emails Security Operations Services when total hosts scanning from a single network class on any port (except ports 445, 135) exceeds 100 hosts.

```
wmd-rule 47 {address-scan group all-hosts class A \
 total-hosts-high 100 total-hosts-low 75} \
 {email-alert to-address securityOperations@sandvine.com}
wmd-rule 48 {address-scan group all-hosts class A dst-port 135}
wmd-rule 49 {address-scan group all-hosts class A dst-port 445}
```

This rule emails Security Operations Services when total address scan bandwidth for a specific network class exceeds 10 mbps.

```
wmd-rule 50 {address-scan group all-hosts class A \
 bit-rate-high 10mbps bit-rate-low 7mbps} \
 {email-alert to-address securityOperations@sandvine.com}
```

This rule emails Security Operations Services when a host has been address scanning for more than 2 hours.

```
wmd-rule 51 {address-scan duration 2hours} \
 {email-alert to-address securityOperations@sandvine.com}
```

This rule emails Security Operations Services when a host from a specific network class is address scanning at more than 500 kbps.

```
wmd-rule 52 {address-scan bit-rate-high 500kbps bit-rate-low \
 250kbps} \
 {email-alert to-address securityOperations@sandvine.com}
```

A.3 Signature Match Mitigation

This rule applies actions only for a specific signature.

```
wmd-rule 100 {signature-match pattern blaster} {...}
```

This rule captive portals any hosts in a specific network class where they are detected as a malicious host. In this case, define an attribute called "infected" and set it in the wmd-rules whose condition you deem indicative of infection.

```
 wmd-rule 101 {signature-match} \
{subscriber-attribute host ?src-ip attribute infected value true}
```

This rule immediately blocks all packets that match a specific signature.

```
detection-config 102 {signature-match-auto-block class all
    \ pattern nachia enable on}
```

Sandvine Incorporated
408 Albert Street
Waterloo, Ontario, Canada
N2L 3V3

Phone: (+1) 519-880-2600
Fax: (+1) 519-884-9892

Web Site: www.sandvine.com