Webview Netflow Reporter

Craig Weinhold (craig.weinhold@cdw.com)

2012-11-12

OVERVIEW

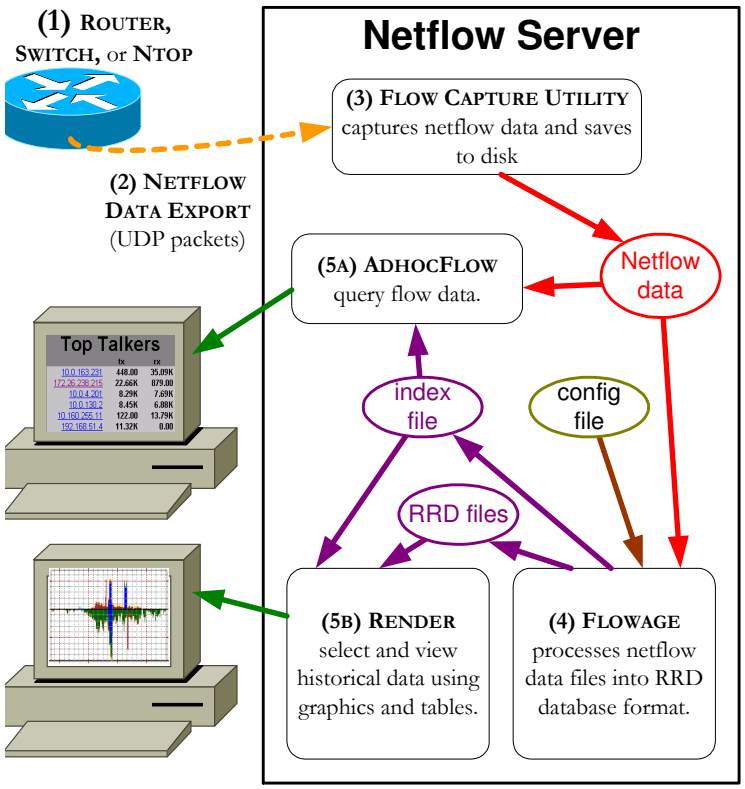
Routers and Layer-3 switches from Cisco and other manufacturers have the ability to catalog IP traffic as a series of *flows*. Each flow is identified by protocol, source and destination IP addresses/ports, type-of-service, and the source interface. Cisco calls this technology *NetFlow*. The data can be exported for use in accounting, usage auditing, capacity planning, etc.

The diagram at right shows how Webview NetFlow Reporter operates. Flows are (1) identified by routers and (2) exported to a server where they are (3) collected and written to disk.

Flowage reads the flow data files and processes them according to its configuration file (4). It maintains the flow data in multiple Round-Robin Databases (RRDs).

There are two client web tools. The NetFlow Traffic Analysis tool (5B, render.cgi) is used to generate graphs and tables of flow data over long periods of time (hours to years). The Ad Hoc Query Tool (5A, *adhoc*.cgi)* enables detailed queries of the raw flow data over shorter time periods (typically 5 minutes to an hour), and is very useful for forensics, spike analysis, top talkers, worm/malware investigation, etc.

Step 4 is where most of the intelligence lies. The configuration file contains Cisco-style ACLs that describe how each flow should be categorized.



Webview relies upon the flow-tools or flowd packages for capturing (3) flows. It relies on the flow-tools package for generating reports (5A). It also relies on the Perl module CFlow to access the files stored by flow-tools.

INSTALLATION

Follow the README and/or INSTALL files contained within the flowage distribution for the most up-to-date details on this process. The information in this section may be stale.

Flowage is composed of the main Perl module 'flowage.pl' and one or more config files (by default there is one named 'flowage.cfg'). Copy these files to the same directory somewhere. Multiple config files would be used to handle multiple flow collections and/or to divide the workload across multiple processes to take advantage of multi-core CPU's.

Flowage has four main data directories - data, watch, temp, and cache. These are described in the section on configuration file format. It's important to create these directories and make sure they are writeable by the UID (user ID) that will be running flowage.pl.

To use the web interface, copy the following files to a location of the web directory structure that allows CGI execution:

```
render.cgi
adhocFlow.html
adhocBlank.html
adhocFlow-help.html
adhocFlow-help.gif
adhocForm.cgi
adhocResults.cgi
adhocIf.cgi
adhocClick.cgi
wvFlowCat.pl
wvFlowCheck.pl
rrdcached-stat.pl
```

There also must be a *graphs* directory that is read/writable by the web user so that render.cgi can create image files. There is optionally an *archive* directory that is also read/writable by the web user to be used as a holding area for capture files.

The web scripts read variables and settings from /etc/webview.conf - this file must exist and be readable by the web user. This file is really a small perl script. It describes the structure of the total webview installation, and is separate from the per-process flowage config files described above.

RUNNING FLOWAGE

There are three ways to run flowage:

1. *Scheduled*. Use cron to run flowage.pl every few minutes. No command line options are needed. Lock files are used to prevent flowage from clobbering a still-running previous instance. This is the preferred way of operation.

2. *Infinite*. Use 'flowage.pl -wait' from the command line to put flowage.pl into an infinite loop, processing files as they arrive.

3. *Config check*. Use 'flowage.pl --check' to check the syntax of the config file and quit.

4. *Test*. Use 'flowage.pl <filename|directory>' to process a single file or directory of files and quit. Processing the file in test mode will not update RRD files, but it will give you an indication of system readiness and performance.

Command-line switches:

--debug
May help troubleshooting (or lead to more confusion). Use if encountering problems.

-f <config file>
Specify a different configuration file. The default is flowage.cfg in the same directory as flowage.pl.

--wait
Run in an infinite loop (see above).

--check
Verify the configuration file syntax and quit.

--nosnmp
Use this switch to suppress SNMP polling.

--daemon <rrdcached>
If using rrdcached service, specify this command line option or set the RRCACHED_ADDRESS environment variable.

By default, log information is written to flowage.log. Watch this file for error messages and performance information. Messages prior to loading of the configuration file will be displayed to the console (STDOUT).

CONFIGURATION FILE FORMAT

Whitespace is ignored and # or ; can be used to add single-line comments. Names are restricted to alphanumerics and underscores (hyphens are not allowed!). Filenames can also include periods and slashes.

Commands that require an arbitrary list of items can wrap across multiple lines (e.g., group, matrix, datafile, host-list and localnexthop) so long as the additional lines are indented and there are no blank or comment-only lines. ACLs appear in Cisco "named" format with one permit/deny per line. All other commands must stay on the same line.

System commands:

include <filename>
Reads another configuration file. This allows nesting or sharing of common configuration settings.

directory data <directory>
Set the root directory of where output and log files are stored. The default is the directory where flowage.pl was started from (script variable $dataDir). This directory and its subdirectories may contain hundreds or thousands of data files. Allow ample disk space.

directory watch <directory>
Set the directory of where flow files are to be found. The default is **/home/flows**. Flowage tracks the directory's contents and process new files when they appear. Use this method if there is some other process that is responsible for archiving/deleting flow files, such as flow-tools. This is the preferred mode of operation when using the flow-tools or flowd collectors.

directory temp <directory>
Set the directory where flowage state files are stored, such as the lock file and dynamic ACL files. The default is the directory where flowage.pl was started from (script variable $stateDir). Although multiple flowage instances can share the watch, data, and cache directories, they ***must*** each have a unique temp directory.

directory cache <directory>
Set the directory where interface information is cached. The default is to use the same directory as **temp**. (script variable $ifCacheDir). If you run multiple instances of flowage (each with its own temp directory), you probably want to use a shared cache directory for all the interface information.

directory flows <directory> *obsolete - use **watch** instead*
An alternative to **directory watch**, this also set the directory where flow files are to be found but it treats the directory as a *drop* location. After processing, each flow file is moved to the **directory save** location or is deleted. Use this method if there is no other process for archiving/deleting flow files, such as with the cflow collector.

directory save <directory> *obsolete - use **watch** instead*
Set the directory where flow files are moved after processing using the **directory flows** method. By default, this variable is undefined causing flow files to be deleted after processing (script variable $saveDir). This is unused if the **directory watch** method is used.

directory hierarchical
By default, flowage stores files directly into the top level of the data directory. With hierarchical turned on, subdirectories are created for each matrix or group component of the RRD filename. E.g., the file **$dataDir/flows.Serial2.HTTP.rrd** would be written as

$dataDir/flows/Serial2/HTTP.rrd. This can make the directory structure more manageable when there are many files/interfaces/datapoints.

period <seconds>

Specifies the amount of time contained within each flow file. The period must be evenly divisible by 60 seconds. It is set in harmony with flow-capture's '**-n *rotations***' option ("the number of times flow-capture will create a new file per day."). E.g., the default period of 300 seconds requires that flow-capture be started with '**-n 288**' so that flow files are stored every 300 seconds.

RRD files created by flowage.pl default to using the period as the step value unless the **Step=<seconds>** and/or **Bucket=<seconds>** options are used (see the datafile section below).

fork <processes>

This enables ***integrated multiprocessing*** with the specified number of worker processes. Each process reads every flow file, but only does deep analysis on a portion of the data. The flows are divvied up according to a hash function:

```
if-matrix           hash on the exporter IP address
nexthop-matrix      hash on the next-hop IP address
protocol-matrix     hash on the IP protocol (tcp, udp, icmp, etc)
```

Integrated multiprocessing does not work with other matrix types, nor with configuration files or datafile definitions that contain different matrix types. The efficiency of integrated multiprocessing depends on how many hash values there are. E.g., an multiprocessing an if-matrix would add no value if there was just a single netflow source.

It is not wise to specify a larger number of processes than there are cores/vCPU's available to the operating system.

This method is easier-to-configure but less efficient than ***explicit multiprocessing***, a technique where flowage.pl is invoked multiple times with different configuration files that are carefully constructed to process chunks of workload (different data files, different flow capture files, etc).

click [loose | strict | strict-other]

Controls how the ad hoc query tool implements ACLs when a graph is clicked on. Loose ACLs are processed independently, while strict ACLs preserve the ACL's placement within a group. Take the following example:

```
ip access-list extended WEB1
 permit tcp any host 10.10.10.10 eq 80 reverse

ip access-list extended WEB2
 permit tcp any host 10.20.20.20 eq 80 reverse

ip access-list extended HTTP
 permit tcp any any eq 80 reverse

group services WEB1 WEB2 HTTP
```

For graphing, flowage.pl checks WEB1 first, WEB2 second, and HTTP third. If none matches, the flow is added to the 'Other' category. When a user clicks on a graph, the ACL used by the ad hoc query tool is determined by the **click** setting:

A setting of **loose** causes the named ACL to be used outside of its group context. E.g., a report of HTTP would include flows that also match WEB1 or WEB2.

A setting of **strict** preserves the location of the ACL in the group. This requires more CPU but the reports are guaranteed to match the graphs exactly. E.g., the HTTP ACL would effectively be changed into this ACL:

```
ip access-list extended HTTP
 deny tcp any host 10.10.10.10 eq 80 reverse
 deny tcp any host 10.20.20.20 eq 80 reverse
 permit tcp any any eq 80 reverse
```

The default setting of **strict-other** behaves like **loose** for all ACLs except **Other**. The **Other** category is strictly interpreted. For example, the "Other" ACL would be:

```
ip access-list extended Other
 deny tcp any host 10.10.10.10 eq 80 reverse
 deny tcp any host 10.20.20.20 eq 80 reverse
 deny tcp any any eq 80 reverse
 permit ip any any
```

This setting can be changed at any time, so it's fine to start off using **click strict** and change it if performance is suffering.

lockfile <filename>
Sets a different lock file. The default lock file is **/tmp/flowage-<config filename>.lck**. The lock file is used to prevent two copies of Flowage from running concurrently in scheduled mode. If Flowage is interrupted or crashes, the lock file will need to be removed. The monFlows.pl utility can monitor and recover that situation.

logging file <filename>
Sets the file to which log information is written. By default, this variable is undefined so all log messages are sent to STDOUT (the screen or console). If filename isn't a full path name, it is stored in $dataDir. Note that messages of **error** severity are sent to STDOUT as well as to the log file. Also, if flowage is run from an interactive command prompt, log messages are displayed to the screen.

logging level <level>
Sets the level of log detail. The levels are **error**, **info**, **trivia**, and **debug**. The default is **trivia**.

logging size <bytes>
Sets the maximum number of bytes to be stored in the log file. Once a log file exceeds this size, they are renamed <logfilename>.old and a new file is started. As a result, disk space usage will be about twice this setting.

Indexing commands:

index file <filename>
Sets the file to which flowage index information is written. The index file is read by web cgi scripts and contains information on where the RRD files are, how they should be described, what colors to use for each datapoint, and compiled ACL definitions from flowage.cfg. If <filename> isn't a full path name, it is stored in $dataDir.

This and the other index commands are only of value for the web interface.

index color <name> <color>
Defines the color to be used for rendering the named ACL, should that ACL be used as a datapoint in a graph. If not specified, colors are automatically assigned. The color can also be specified with the '**ip access-list extended**' command. The color is specified either with a hexidecimal #RRGGBB value or a color name ("green", "cornflowerblue", etc). For a complete list of color names, look at the %colors hash definition near the end of flowage.pl.

index description <filename> <description>
Provides a verbose description for a datafile that will be displayed in the web interface (see "Data Manipulation Commands" below for how to define a datafile). The filename here must match the filename of the datafile exactly. The description can contain any characters or even html code, provided it fits on a single line. Quotes around the description are optional, but advised.

Network Environment commands:

exporter <name> <ip> [non-cisco] <snmp_directives> *explicit definition*
exporter <auto|snmp|dns> <ip/mask> [non-cisco] <snmp_directives> *exporter auto-discovery*

<snmp_directives> can be
<community [community ...]> [version [port]] [non-cisco] *old style*
<snmp-session [snmp-session ...]> [non-cisco] *new style*

Explicit exporter definition hardcodes an IP address and exporter name together. This also enables the use of interface names in access-lists (e.g., "permit ip Ethernet0/0 any Serial0/0 any exporter 10.10.10.1").

Exporter auto-discovery learns exporters from the flow data and the exporter name from SNMP sysName and/or DNS. Only exporters that fall within the ip/mask will be discovered. The interfaces of discovered exporters cannot be referred to in access-lists.

Both syntaxes let the exporter and interface to be used in interface matrices and with descriptions and aliases in the web interface. If an exporter is seen that is not covered by an **exporter** statement, a generic name like "10-10-10-1#SnmpIf12" is used (unless **exporter no-unknown-exporters** is used to suppress this).

The old SNMP syntax specifies one or more community strings, followed by version and port. This is only compatible with SNMP v1 and v2c. The new SNMP syntax specifies one or more **snmp-session** definitions, which support v1, v2c, and v3 (see the **snmp-session** command below).

If multiple communities or snmp-sessions are defined, they are tried in order on each exporter. When one is found that works, it is used for all further SNMP communication.

Flowage attempts to be SNMP friendly. For example, on devices with a very large number of interfaces, only those with IP addresses are polled. Interface information is cached for 12 hours or until a reboot or interface change is detected. To detect interface changes, the cisco-specific SNMP OID ifTableLastChange is used. A non-cisco exporter should use the **non-cisco** keyword to disable these optimizations.

On Cisco devices, it's highly recommended to configure **snmp-server ifindex persist**.

See the section on SNMP INTERFACE CACHING section for more details.

exporter no-unknown-exporters

This will suppress the creation of RRD files for exporters whose names are not known, either by explicit definition or by auto-discovery.

This helps flowage remain stable even if bombarded with bogus/forged NetFlow data.

snmp-session <name> parameter1=value1 parameter2=value2 ...

An snmp-session is a collection of SNMP parameters to use. The principal ones are:

parameter	value	notes
version	1, 2, or 3	default 1
community	*user-supplied*	v2 only, defaults to 'public'
username	*user-supplied*	v3
authpassword	MD5, SHA-1	v3
authprotocol	*user-supplied*	v3
privpassword	DES, 3DES, AES	v3
privprotocol	*user-supplied*	v3

Lesser-used parameters include port, localaddr, localport, timeout, retries, maxmsgsize, privkey, and authkey. Refer to the Net::SNMP perl module documentation for descriptions of those parameters.

localnexthop <ip [ip [...]]>

Specify one or more local next-hop IP addresses. Flows with next-hops from this list are ignored. For instance, take the network:

```
  e0 10.0.1.0/24
R1
 |s0 10.0.0.1/30
 |
 |s0 10.0.0.2/30
R2
  e0 10.0.2.0/24
```

If both R1 and R2 export NetFlow, then flows from 10.0.1.x to 10.0.2.x will be exported twice and could lead to double-counting. By setting **localnexthop 10.0.0.1 10.0.0.2**, duplicate flows will be ignored.

Local next-hop checking is not needed for interface or next-hop matrices (***if-matrix*** or ***nexthop-matrix***, discussed later). In those cases, flows are tabulated in a manner that avoids the problem of double-counting.

Note that the *next-hop IP address* is only consistently available when NetFlow is collected from routers/switches. External probes and softflowd will not populate this field.

dns servers <ip [ip [...]]>
Specify one or more DNS servers to use for DNS queries. The default is to use the DNS servers configured on the WebView server itself. See the section on 'DNS' for more information. These settings only affect flowage.pl operation. DNS settings for the web cgi scripts are stored in /etc/webview.conf.

dns timeout <seconds>
Specifies how long Flowage will wait with each bulk DNS lookup. The default is 5 seconds. See the section on 'DNS' for more information.

dns pacing <integer>
Specify how many DNS queries are sent before answers are polled during the bulk DNS lookups. The default is 10. This may need to be lowered if Flowage is having difficulty getting DNS responses.

Data Manipulation commands:

ip access-list extended <name> [append] [color <color>]
This defines an access control list (ACL) that will match certain flows by their IP addresses, port numbers, protocols, size, etc. The actual body of the ACL is described in detail in its own section later on. Terse names are preferable because the ACL name will be incorporated into the data file name. The optional **color** is a shortcut method of defining an '**index color**' for the ACL. The optional **append** keyword adds new rules to a previously existing ACL of the same name, which can be useful when merging ACL's from multiple files.

Lines of an access-list can be an arbitrarily complex list of permits and denys.

ip host-list @<name> <ip|subnet|@name [ip|subnet|@name [...]]>
Defines a static host-list and fills it with IP addresses, subnets, and/or other host-lists. These lists can be used in ACLs or datafile definitions in place of IP addresses or subnets. Internally, host-lists are stored in a Trie structure that can handle many thousands of entries without a performance penalty.

One benefit of static host-lists is that their use can make some ACLs shorter and more efficient. For instance,

```
ip access-list extended matchServers
 permit ip any host 10.43.4.14 reverse
 permit ip any host 10.98.5.3 reverse
 permit ip any 10.53.49.0 0.0.0.255 reverse
```

could be written as:

```
ip host-list @serverHosts 10.43.4.14 10.98.5.3 10.53.49.0/24

ip access-list extended matchServers
 permit ip any host @serverHosts reverse
```

ip access-list map <name>
An ACL map is an alternative to extended ACLs that can be tremendously more efficient in situations when classification can be done by IP address alone. Consider this example:

```
ip access-list map myAclMap
 Citrix    @citrixServers
 Proxy     10.43.15.0/24
 B2BProxy  10.43.15.23
 Voice     @voiceServers @voiceGateways
```

Each line of an ACL map is a traffic category followed by one or more host-lists, IP addresses, or subnets. Any flows that match one of the IP/subnets are placed in the traffic category. The order of the categories within an ACL map has no effect. If IP/subnets overlap, the most-specific wins (e.g., B2BProxy is more specific than Proxy in the sample above).

One concern is when the flow source and destination both match categories (e.g., a flow from a @citrixServer to a @voiceServers in the sample above). When this happens, flowage

prefers the one with the TCP/UDP port number under 1024 or, if both TCP/UDP ports are high, the flow source. To minimize the risk of this happening, define the IP/subnets are narrowly as possible.

The chief downside of ACL maps is that they do not support TCP/UDP ports or other more advanced flow attributes. They can, however, be used in conjunction with traditional ACLs (see **'group'** below).

A powerful secondary use of access-list maps is to improve the readability of the adhoc reporting engine by adding a column next to each IP showing its category. All that is needed is to define an ACL map in flowage.cfg; it is not necessary to attach the ACL map to a datafile for processing.

dynamic <acl> [source-ip @<name>] [destination-ip @<name>] [flow %<name>] [timeout <seconds>]

Any flow that matches the given ACL will have portions of that flow remembered in a host-list or flow-list structure for a period of time, where it can be referred to by other ACLs.

If **source-ip** and/or **destination-ip** are specified, the matching flow's source and/or destination IP are added to the named host-lists (i.e., **@NAME**). Host-lists can be used in place of a host IP address in another ACL. E.g., **permit tcp host @NAME any**. Below is a example of this usage:

```
ip access-list extended FTP_control_in
 permit tcp any any eq 20

ip access-list extended FTP_control_out
 permit tcp any eq 20 any

# Define two dynamic ACL's that maintain the host-lists @FTPUsers and @FTPServers
dynamic FTP_control_in source-ip @FTPUsers destination-ip @FTPServers timeout 60
dynamic FTP_control_out source-ip @FTPServers destination-ip @FTPUsers timeout 60

ip access-list extended FTP
 permit tcp any any eq 20 reverse
 permit tcp any any eq 21 reverse
 permit tcp host @FTPUsers gt 1023 host @FTPServers gt 1023 reverse
```

If **flow** is specified, the matching flow is added to the named flow-list (e.g., **%NAME**). ACL entries with a **flow %NAME** clause will succeed if all of the fields referenced in the ACL entry match a flow from the flow-list. Fields may include:

$srcip	source IP address
$srcport	source TCP/UDP port number
$srcas	source BGP autonomous system number (AS)
$srcif	source interface
$dstip	destination IP address
$dstport	destination TCP/UDP port number
$dstas	destination BGP autonomous system number (AS)
$dstif	destination interface
$exporterip	exporter IP address
$nexthopip	next-hop IP address
$dscp	dscp (highest 6 bits of ToS)
$precedence	IP precedence (highest 3 bits of ToS)
$tos	type-of-service (entire ToS byte)

Here is the same FTP example redone for flow-lists:

```
ip access-list extended FTP_control
 permit tcp any any eq 20 reverse

# Define two dynamic ACL's that maintain the host-lists @FTPUsers and @FTPServers
dynamic FTP_control flow %FTPSessions timeout 60

ip access-list extended FTP
 permit tcp any any eq 20 reverse
 permit tcp any any eq 21 reverse
 permit tcp host $srcip gt 1023 host $dstip gt 1023 flow %FTPSessions reverse
```

The dynamic constructs are powerful for heuristic interpretation of flows, but there are caveats.

1. Flow capture files are only roughly chronological. Even with careful time sync on your exporters, flowage may encounter an FTP data flow before its FTP control flow. For more precision, use flow-tools' flow-merge utility to groom the capture files before processing them with flowage.
2. The dynamic ACL is run against every flow, so it should be made as efficient as possible.
3. Flow-lists are less efficient than host-lists so use them sparingly. (internally, a flow-list is just a perl hash function, while a host-list is an efficient Trie structure).
4. The ad hoc query tool does not process dynamic ACL logic. E.g., clicking on a graph of a category that uses a dynamic ACL will only report on the non-dynamic components of each traffic category.

group <name> <acl|map [acl|map [...]]>

ACLs and ACL maps can be grouped together in an ordered list. For example, a group named 'Applications' could contain the ACLs 'FTP', 'Email', and 'Microsoft'. When the group 'Applications' is used to define a datafile (see below), the result will be five separate keys: the three ACLs plus two special values, 'Other' which matches traffic other than the grouped ACLs and 'Any' which matches traffic in any of the ACLs. The 'Any' category is not tracked when a group of ACLs is combined with a matrix (see below).

A group of ACLs/ACL maps is processed in the order listed in the config file. I.e., if a flow matches the first ACL/ACL map in the group, the others won't be tested. That means a group is *exclusive* and no flow will match more than one ACL/ACL map of a group.

Groups are one of the two principle ways for defining functionality for the render.cgi web script.

(if|as|ip|port|subnet|nexthop|protocol)-matrix <name> [auto [counter] [mask=<filter>] [bits=<number>]] [xform=<transformation> ..] [aliases=<file>|simple] [descriptions=<file>] [authorization=<file>][value ..]

The matrix commands define groups whose keys are IP addresses, subnets, port numbers, protocols, etc. These matrices can be prepopulated with keys. If so, all unmatched traffic goes into an 'Other' category. Or, with the **auto** keyword, the matrix will automatically create new keys as they are found in the NetFlow data. The **mask** option defines a filter to restrict the range of keys eligible to be automatically added. The filter is a comma-separated list of values in either IP/bits or integer range format, depending on the type of matrix. The **bits** option sets a maximum netmask length for automatically added subnet matrices. The **xform** option transforms AS numbers.

With the **counter** option, flowage records a count of the number of unique keys seen without creating a separate data files for each key. This is useful for tracking a count of unique values (AS's or IP addresses), without recording what the values actually are.

The **aliases**, **descriptions**, and **authorization** options point to files that provide information to help label and control access to matrices from the web interface. Using **aliases**=**simple** will creates one alias for every description known for the matrix (e.g, interface descriptions learned automatically or subnet descriptions learned from a file).

Matrices are the second of the two principle ways for defining functionality for the render.cgi web script. They are described in more detail in their own section below.

datafile [rrd|packed-rrd|text|csv|flow] <filename> <acl|group|matrix [acl|group|matrix [...]]> [MaxSize=<size>] [Step=<seconds>] [Bucket=<seconds>] [Window=<seconds>[:<seconds>] [Consolidation=<stuff>] [Timestamp=<method>] [IPTracking] [<category:directory>] [in=<acl>] [out=<acl>]

Defines an output file for traffic that matches *all* ACLs, groups, and/or matrices listed. An ACL or group can be preceded by a '!' to permit only non-matching flows.

File types include:

'**rrd**' is a round-robin database that records input/output byte, packet, flow, and, optionally, IP counters at regular intervals (typically 1 or 5 minutes). RRD files are manipulated with RRDTOOL, a powerful open-source data storage and retrieval package. RRD is the data format used by render.cgi to create graphs.

'**packed-rrd**' is an RRD file that stores multiple datapoints in each file. A standard flowage RRD file has just one datapoint per file, which has the advantage of making it

easy to add/remove datapoints. A 'packed-rrd' file, on the other hand, contains all datapoints for the last *group* of ACLs in the datafile definition (e.g., traffic categories like 'Email', 'Internet', etc). This significantly reduces the number of RRD files and can improve disk performance, but the number of datapoints is set in stone and cannot be changed (a savvy rrdtool user may be able to craft a workaround).

'csv' is a flat-file in comma-separate value format. It can be used by render.cgi, imported to Excel, or processed by other scripts. ***Rarely used.***

'text' is a flat-file pre-formatted in columns for easy reading/printing. Lines will be wider than 80 columns. ***Rarely used.***

'flow' is raw Cflow format, readable by other applications. ***Rarely used.***

The filename can include slashes to specify root subdirectories, if desired (e.g., 'summary/flows').

File size limits for text, csv, and flow files

'MaxSize=<size>' can be included for text, csv, and flow files to limit the file size. When the file grows larger than this number (in bytes), it is copied to *.old and is reset. An existing *.old file will be overwritten. Thus, the disk usage may be roughly twice MaxSize.

Steps, Buckets, Windows, Timestamps, Consolidations, and Periods for RRD files

'Step=<seconds>' specifies the duration of a datapoint in the RRD file, typically 60 or 300 seconds. Once an RRD file is created, the step cannot be changed without deleting all the existing RRD files first. Any step from 1 second to the **period** variable can be used. By default, step equals **period** and timestamps of each flow are derived from the timestamp embedded in the filename of the raw flow file. A step value less than **period** will use the timestamps embedded within the flows which makes it mandatory that the exporting device have a reliable Network Time Protocol (NTP) configuration.

A note on Step and and the flow exporter's *"active flow timeout"*

With traditional Cisco Netflow ("ip route-cache flow", "ip flow ingress", and/or "ip flow egress"), the minimum active flow timeout is 1 minute. With this configuration, experience has shown that setting the step or bucket size to less than 60 seconds doesn't gain much in the way of accuracy for *most* enterprise traffic patterns (one exception would be an unusual network with many short, bursty flows).

With flexible Netflow ("ip flow monitor"), the minimum active flow timeout is 1 second. If you plan to create RRD datafiles with a 1 second step, pay close attention to the disk/CPU burden. A good approach would be to only store the high resolution data for a day or less and to use 5-minute averages for longer ranges. E.g., Consolidation=0.25:14:90:720 to store 6 hours of **step** resolution. You can also use **buckets** (described below) with high resolutions (e.g., bucket=10 & step=30/60).

'Window=<seconds>[:<seconds>]' sets the past/future bounds of the window of valid flow starting timestamps. By default this window is 450 seconds into the past and 450 seconds into the future (1.5 * period), where the "current" time is derived from the highest start time of all flows processed. The Window parameter rarely needs adjustment unless the NetFlow exporters are using an active flow timeout greater than 5 minutes.

Flows with starting timestamps outside this window are dropped and reported as "no-bucket flows" in the log. If too many no-bucket flows are seen, the window is automatically reset. Chronic reports of no-bucket flows should be investigated and are often due to inconsistent or unreliable Network Time Protocol (NTP) on the NetFlow exporters.

For example, a Cisco router has a default 30 minute active flow timeout (generally the timeout should be set down to 1 minute, but that's often overlooked). To properly tabulate all flows from such a router, 'Window=1850' should be configured on the datafile.

Another way to think about it is that datapoints won't be available for graphing until they are outside of the window. E.g., with a window of 1850 seconds, a NetFlow graph will always lag about 30-35 minutes behind current time. With default settings, the lag time is only 5-10 minutes.

'Timestamp=<method>' specifies a method to assigning flows to each time interval. For example, say step=60 and a 400 KB flow is received that starts at 10:15:58 and ends at 10:18:35. The four methods are:

start	the entire flow is tabulated at the step that represents its start time (400 KB @ 10:15)
end	… its end time (400 KB @ 10:18)
average	… the midpoint of the flow (400 KB @ 10:17)
distribute	the flow is distributed across steps from its start to end time (100 KB @ 10:15, 10:16, 10:17, and 10:18)

The default setting is **distribute** which handles most situations well. If step/bucket is set equal or greater than the exporting router's flow active timeout, then you can safely use **Timestamp=average** and see a slight performance increase.

'Consolidation=<step:5-minute:1-hour:1-day>[:/regexp/]' determines how much history the RRD file should keep for each sample size[1]. The default is **"2:14:90:730"** which reads "keep 2 days of **step** samples, 14 days of 5-minute average/max samples, 90 days of 1-hour average/max samples, and 730 days of 1-day average/max samples." The default consolidation creates datapoint files that are 900KB for the default step=300 and 3MB for step=60.

Each history value can be less than a day (e.g., "0.5" for 12 hours) or zero.

The benefit of consolidation is that graphs can go back months or years without needing to store and process every individual data point. E.g., to draw a 1-month graph, rrdtool need only process 750 one-hour samples rather than 9,000 five-minute or 45,000 one-minute samples.

The drawback of consolidation is that precision reports are not available outside of the available history. E.g., if rrdtool was drawing a 8-hour graph from today it could use precise **step** or 5-minute samples. But if it drew the same graph from 3 weeks ago then it would only have coarse 1-hour samples. Consolidation is optional and one could create a file with "365:0:0:0" to keep **step** resolution for an entire year.

Note: prior to flowage 2.07, 5-minute samples weren't stored. If an old-style config syntax is found (e.g., "14:90:730" or "MAX:14:90:730"), then it is converted to "14:0:90:730").

Datapoints can use different consolidation functions by means of a regular expression match against the full RRD filename. For example, a typical interface-matrix/category datafile will create datapoint files like these:

```
/var/log/webview/flows/data/Summary/vpn-rt1#FastEthernet0%2f0/Internet.rrd
/var/log/webview/flows/data/Summary/wan-rt2#Serial0%2f0%2f1%3a5/Email.rrd
```

(note that characters like '/', ':', and '.' are url-encoded as %2f, %3a, and %2e - e.g., that last file actually reads "Serial0/0/1.5")

Here are some sample consolidation functions

```
Consolidation=7:30::365:/vpn-router-1#FastEthernet0%2f0/    # specific interface
Consolidation=2:14:90:730:/Ethernet/                        # interface type
Consolidation=0.5:14:90:730:/Email/                         # category name
```

A handy use is to set better resolution for WAN interfaces than for LAN interfaces.

```
Consolidation=35:40:90:730:/(Serial|ATM|POS|MFR|Multilink)/ # WAN interfaces
Consolidation=0.25:14:90:730                                # everything else
```

Note that a graph containing datapoints with different consolidation functions may look unusual - with some datapoints using more precise samples and others using more coarse values.

[1] For those familiar with RRD file formats, these numbers are used to set the consolidation function [CF] and number of rows for **step**, 5-minute, 1-hour, and 1-day resolutions. Webview automatically carries forward both AVERAGE and MAX consolidated data.

'Bucket=<seconds>' ***Rarely used!*** This enables the use of the bucket algorithm and specifies the size of the bucket. The bucket size should be evenly divisible into the step size, or it will be rounded. With the bucket algorithm, each datapoint of the RRD file will take on the single largest bucket from within the step. This is called a *high water mark*. For example, say Step=300 and Bucket=60 and the following byte counts are recorded over a 300-second step:

	start time	duration	bytes	effective Mbps
bucket 1	10:10:00	60	20,000,000	2.7
bucket 2	10:11:00	60	30,000,000	4.0
bucket 3	10:12:00	60	50,000,000	6.7
bucket 4	10:13:00	60	90,000,000	12.0
bucket 5	10:14:00	60	40,000,000	5.3
step	10:00:00	300	230,000,000	6.1

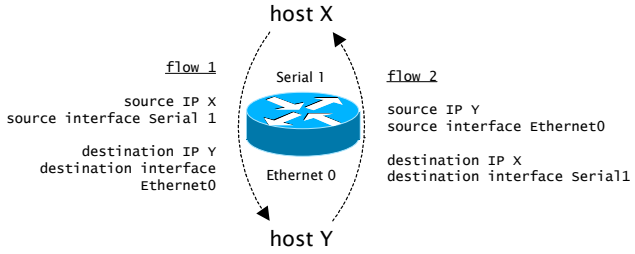
Without the bucket algorithm, the datapoint would be recorded as a strict average of 6.1 Mbps. With the bucket algorithm, the datapoint would be recorded as 12 Mbps, based on the 10:13:00 bucket.

The benefit of the bucket algorithm is that traffic spikes can be recorded without the disk and CPU burden required to store all samples at such a high resolution.

The downside of buckets is that the reported data is no longer a strict average, which can lead to confusion with understanding how the data is reported since there will be slightly skewed graphs and elevated averages/peaks.

Tracking of IP addresses in RRD files

'IPTracking' enables tracking of unique source and destination IP addresses for each RRD datapoint. Without the command, IP tracking is disabled. Tracking uses extra CPU, but can be very useful in deriving per IP data, such as application bandwidth per user. To understand tracking better, consider two flows being tabulated using an interface matrix:



Each flow's source and destination IP are tabulated on only the source interface of the flow. I.e., tabulation of flow 1 is on Serial 1 and flow 2 is on Ethernet 0. This allows the render web interface to behave consistently (if not always intuitively). E.g., a user might want to analyze traffic from Serial 1 by source IP ("user") or destination IP ("server").

Input/output for RRD files

Each RRD datafile stores flows as either 'input' or 'output'. However, determining the direction of a flow can require some guidance (e.g., is it "in from the server" or "out to the Internet"). Here is how flowage handles direction:

- Matrices based on keys that are inherently directional (i.e., IP, subnet, AS, port, and interface) are tabulated both as input from the "source" and output to the "destination", making graphs of individual elements very intuitive. However, since each flow is tabulated twice, this can lead to confusion when graphing an aggregate of multiple entities. For example, a graph that combines all interfaces of a router will report double the amount of traffic.

- Matrices based on only nondirectional keys (i.e., protocol and nexthop) are tabulated once in the input category.

- ACLs (and Groups of ACLs) that are not combined with matrices may be directional or directionless. It's impossible for Flowage to guess at the operator's sense of direction, so the default is to tabulate everything as 'input'. However, the RRD datafile definition can override this by including **'in=<acl>'** and/or **'out=<acl>'**. These **'in'** and **'out'** keywords specify an ACL to be used to determine the flow's

direction. Just one keyword acts as a Boolean test ("in *or* out"). With both, flows not matching either ACL are dropped ("in *or* out *or* bitbucket").

- The **'in=<acl>'** and **'out=<acl>'** syntax can be configured on datafiles that use interface matrix, but they will only take affect if a flow's input and output interface are both 0 (null). This may be the case when the flow is generated by a probe or tap.

ACLs, Groups, and Matrices in datafiles

Groups and matrices are a powerful way of using flowage, and the companion web script render.cgi relies on them heavily. By specifying one or more groups, a datafile is created for each element of the each group (and the automatic category 'Other'). A matrix creates one datafile for every IP, port, subnet, AS, protocol, interface, or nexthop IP either predefined in the config file or observed in the NetFlow data (with the matrix definition **'auto'**).

Practically speaking, a datafile should only have two groups or matrices (e.g., matrix **interface** and group **categories**), though there are a few rare use cases for more (e.g., matrix **source-subnet**, matrix **destination-subnet**, group **categories**).

An undisciplined use of groups and matrices can create hundreds or thousands of RRD datafiles in the $dataDir directory, making it important to monitor disk space and to use ACLs to restrict the scope of what matrix values can be automatically created (this is especially true for IP and port matrices). Also, the sheer number of files in a single directory may be a problem on some systems. If this is a concern, use the **directory hierarchical** global command to store files in subdirectories of $dataDir.

The option **<category:directory>** causes raw flows matching the named category to be written to a directory using the same file name as is being read. This works only when matrices and groups are used together – in other situations one should use a separate **datafile flow** definition to export raw flow data. The primary use of this is to split raw flows out to separate files to make further processing easier. ***This is an abandoned feature that may not work.***

MATRICES

Below is an explanation of each of the matrix types:

if-matrix <name> auto [counter] [aliases=<file>|simple] [authorization=<file>]
'if' is shorthand for **'interface'**. The key is the exporter's name or IP address and the interface name or snmp index, joined by a number sign. These keys are most readable if an **exporter** is defined or auto-exporter learning is enabled. ("**WanRouter1#FastEthernet1/0**" is more descriptive than "**10.45.23.9#snmpIf3**").

Currently there is no way to prepopulate specific interfaces so the **auto** keyword is mandatory.

There is no option for specifying a **descriptions** file since descriptions are loaded automatically from the router if an exporter is defined.

as-matrix <name> [auto [counter] [mask=<filter>]] [xform=<transformation> ..]
[descriptions=<file>] [aliases=<file>|simple] [authorization=<file>] [as ..]
'as' is an short for **'autonomous system'** (also abbreviated as AS), a 16-bit integer that identifies a service provider within the BGP routing protocol. NetFlow v5 can export the AS numbers of either the immediately adjacent BGP neighbor (the *peer*) or the ultimate source/destination AS (the *origin*) depending on how the router is configured:

ip flow-export version 5 peer-as
or
ip flow-export version 5 origin-as

The matrix key is simply the integer AS number. The use of **auto** with *origin-as* on a router with full Internet BGP tables will lead to one file being created for each of the 45,000+ AS's.

The xform option converts an AS to a new value. For example, many routers report 0 for the local AS. The option **xform=3599:0** would convert AS0 back to a real AS. Another use is to aggregate AS numbers for reporting simplicity. For example, the following will convert all 4000+ APNIC-assigned AS's (which approximates the Asia Pacific region) into the single private AS64512:

```
xform=64512:4608-4764,7467-7722,9216-10239,17408-18431,23552-24575,
    37888-38911,45056-46079
```

A good source of information on BGP AS numbers is www.cidr-report.org.

**port-matrix <name> [auto [counter] [mask=<filter>]] [descriptions=<file>]
[aliases=<file>|simple] [authorization=<file>] [port [port […]]**

A port number is a 16-bit integer that applies to TCP and UDP traffic only. When prepopulating values, specify the integer port number (no names).

Like IPs, an **auto** port number matrix can also generate many files unless restricted by adding another ACL. For instance, it might be wise to define an ACL that limits port numbers to the range of 1..1023, and then couple this ACL with an **auto** port matrix. For protocols other than TCP and UDP, the port number is 0. The mask can also restrict the range of **auto** port numbers.

**ip-matrix <name> [auto [counter] [mask=<filter>]] [descriptions=<dns|file>]
[aliases=<file>|simple] [authorization=<file>] [ip|@host-list […]]**

If the **auto** keyword is used, IP addresses are taken from the flow data. Using **auto** with an IP matrix will create a very large number of files unless the IP range is restricted using one of two methods: (1) an ACL that restricts the amount of flow data to a specific router, host, etc, or (2) by specifying a mask to limit the IP matrix to a specific range, such as '10.23.50.0/24'.

For IP and next-hop matrices, **descriptions=dns** instructs flowage to maintain a cache of reverse DNS entries (in-addr.arpa) to be used as descriptions for each IP address.

The matrix key takes the form **10-96-48-0**, which is kind of goofy but necessary because of filename standards. When prepopulating values, use the standard syntax (e.g., **192.168.13.15** or **10.0.0.1**).

**subnet-matrix <name> [auto [counter] [mask=<filter>] [bits=<subnet bits>]
[exact=<subnet bits>]] [descriptions=<file>] [aliases=<file>|simple]
[authorization=<file>] [subnet|@host-list […]]**

If the **auto** keyword is used, subnets are identified by using route information embedded within the NetFlow V5 export packets. All routers can include the netmask info in their export. However, the **auto** keyword is not suitable for NetFlow collected from some switches and external NetFlow probes that do not populate the route mask fields (unless **bits** is also specified)

If **mask** is specified, only subnets that fall within the mask will be added (e.g., “10.32.0.0/11”). If **bits** is specified, it will enforce a maximum netmask length, rounding down any smaller routes (e.g., **bits=24** will round /29 or /30 routes to the appropriate /24). Also, **bits** will be used as a subnet mask for flows that do not have routing information or follow the default route. If **exact** is specified, all subnets are defined according that bit value regardless of embedded routing information.

Without the **auto** keyword, subnets are manually specified in the configuration file, either by listing them one by one or referring to one or more @host-list structure(s).

The source or destination of a flow will be tabulated under only a single subnet. If overlapping subnets are available, flowage always uses the most specific.

The matrix key takes the form **10-96-48-0_24**, which is kind of goofy but necessary because of filename standards. When prepopulating values, use the standard syntax (e.g., **192.168.13.0/24** or **10.0.0.0**).

**nexthop-matrix <name> [auto [counter] [mask=<filter>]] [descriptions=<dns|file>]
[aliases=<file>|simple] [authorization=<file>] [ip [ip […]]**

A matrix can be composed of the IP addresses to which the exporting router forwards the flow (aka the *upstream* router). If the flow terminates on an interface directly attached to the exporting router, the nexthop value is the network number of the interface. The nexthop value is 0.0.0.0 for multicast and broadcast traffic.

This makes a good **auto** matrix since the number of nexthops should be small (hopefully).

For IP and next-hop matrices, **descriptions=dns** instructs flowage to maintain a cache of reverse DNS entries (in-addr.arpa) to be used as descriptions for each IP address.

protocol-matrix <name> [auto [counter] [mask=<filter>]] [descriptions=<file>] [authorization=<file>] [aliases=<file>|simple] [protocol [protocol […]]

In an IP packet, the protocol is 'tcp', 'udp', 'icmp', 'gre', etc. There are relatively few of these and they can be specified by name or number.

Settings common to all matrices

The **mask** keyword sets a filter which limits the keys added by the **auto** keyword. Multiple filters can be combined with commas, but be sure not to have any whitespace. The format of the filter varies with the type of matrix:

IP, nexthop, subnet filters	**Port, AS, and protocol filters**
mask=192.168.0.0/16	mask=1-10
mask=10.7.0.0/16,64.73.0.0/24	mask=1-10,23,50

The **counter** keyword causes flowage to track the size of the matrix internally, but instead of creating datafiles for every key, it writes a single file with a count of the number of unique keys. This is useful for tracking a quantity of items without tracking the individual items. For example, this can answer 'how many IPs were seen?' or 'how many unique AS numbers were accessed?'

The **descriptions** and **aliases** keywords point to filenames that provide descriptions/aliases for the matrix for use within the render.cgi web interface. See the next section for details.

The **authorization** keyword specifies a filename that defines users and what portions of a matrix they can have access to within the render.cgi web interface. See the section on matrix security for more details.

MATRIX DESCRIPTIONS AND ALIASES

Overview

Matrix keys are IP addresses, subnets, interfaces, integers, and other pieces of cryptic data that the untrained eye may find unfriendly. The render.cgi web interface has two mechanisms to help with this. A ***description*** is a one-to-one relationship between a matrix key and a human-readable name (multiple keys may have the same description). An ***alias*** can be have a one-to-one or a many-to-one relationship between a human-readable name and matrix keys or their descriptions.

Within the render.cgi interface, the user is presented with a pull-down menu that allows the selection of aliases or raw matrix keys. At the bottom of the screen is an information box that shows the description when a matrix key is highlighted, or the target matrix key(s) when an alias is highlighted. Descriptions themselves only display in this box. To use descriptions in the pull-down menu, they must be converted into aliases with a short aliases file (shown below).

Aliases are useful for having a single name refer to one or more interfaces, IPs, subnets, etc. For instance, 'DMZ VLAN' might be an aggregate of interfaces on several routers. Any graphs of 'DMZ VLAN' would automatically aggregate the interfaces together and show them as a single datapoint.

Descriptions of interface matrices are automatically taken from the router using SNMP if an **exporter** is defined.

Descriptions of IP and nexthop matrices can be learned from DNS if the matrix includes the keyword **descriptions=dns**. See the section on 'DNS' for more information.

Descriptions of other matrices can only be provided through an external file, described in the section on 'File format' below.

Some examples of the relationship between matrix keys, descriptions, and aliases:

matrix	Key	Description	Alias match	Alias target	Result
ip nexthop	10.96.23.5 10.96.23.6 10.96.23.9	www.acme.com wap.acme.com db.acme.com	.*\.acme.com or 10.96.23.0/24	"ACME servers"	ACME servers
subnet	10.96.2.0/24	server farm	10.96.0.0/16	"Building C"	Building C
as	7018 701	AT&T UUnet	701,7018	"Upstream ISPs"	Upstream ISPs
port	23	telnet	1-1024	"Well-known"	Well-known
protocol	17	udp	2-5,7-16,18-255	"Lesser protocols"	Lesser protocols
if	Ethernet1/0 Ethernet1/1	"primary DMZ" "backup DMZ"	.*(DMZ)	$1	DMZ

File format

The general format is similar to the configuration file with respects to whitespace, comments, and multiple lines. Elements with embedded spaces should be single- or double-quoted. As a rule of style, it's best to single-quote regular expressions. The typical form is:

```
TARGET VALUE VALUE VALUE ...
```

In the case of descriptions, TARGET is the human-readable description and VALUE is the matrix key. For aliases, TARGET is the human-readable alias and VALUE is the ***alias match***, a regular expression, subnet, or integer range.

One special case is that if the first element is an integer or IP address, then the file is interpreted as:

```
VALUE TARGET
```

This logic allows flowage to read standard files like **hosts**, **services**, and **protocols**.

Alias matches and targets

The alias matches can be regular expressions, subnets, or integer ranges. The alias target can be a simple text string and can optionally include references to the regular expression match ($1, $2, etc). In addition, a few predefined variables can be used:
if-matrix alias - $f1=router, $f2=interface, $f3=description, $f4=router#interface

Descriptions into Aliases

The special use **aliases=simple** creates one alias for every description. It is equivalent to having an aliases file with this transformation:

```
'$1'       '(.+)'
```

The VALUE/alias match is '(.+)', which is a regular expression that matches the entire description, so long as it is at least one character long.

The TARGET is '$1', which is replaced by the characters matched in parenthesis in the VALUE/alias match. In this case, the '$1' will be replaced by the entire description.

MATRIX AUTHORIZATION AND SECURITY

Within the render.cgi web interface, it may be desirable to restrict which matrix values a certain user has access to. For instance, a branch manager may be allowed to see only subnets within his/her branch. Render.cgi uses an authorization scheme based on environment variables, such as those set by standard web server CGI (Common Gateway Interface):

```
REMOTE_ADDR    The IP address of the web user
REMOTE_NAME    The user name as authenticated by the web server
```

The file format is similar to alias and description files,

```
TARGET VALUE VALUE VALUE ...
```

but the TARGET clause is composed of an environment variable, an equal sign, and a match string (e.g., REMOTE_ADDR=10.5.2.13). If a matrix value matches any of the VALUE clauses, the user is authorized to view it. All matches can be regular expressions, subnets, or integer ranges. For regular expressions in matrix matches, both the matrix item's description and the matrix value are checked.

The first matching TARGET clause is used. If no matching TARGET clauses are found, the user will get a security error:

You aren't authorized for any variables of type 'Interfaces'

Match clauses for interface-matrices can include the exporter and/or the interface. To match both, use the 'exporter#interface' syntax. For example, 'WRTR-09#FastEthernet4/0/0' matches interface 'FastEthernet4/0/0' on exporter 'WRTR-09'.

Two examples authorization files are included later in this documentation.

DATAFILE GENERAL GUIDELINES FOR GROUPS AND MATRICES

Example #1 When defining a datafile, it's very typical to combine a matrix with a group of ACLs. For example, a matrix of interfaces can be combined with a group of **Services** (HTTP, Email, Citrix, etc) for a good view of what traffic is traversing each link in the network:

```
# define ACLs for each traffic category...

ip access-list extended HTTP
 permit tcp any any eq 80 reverse    # HTTP
 permit tcp any any eq 443 reverse   # HTTPS

ip access-list extended Email
 permit tcp any any eq 25 reverse    # SMTP
 permit tcp any any eq 143 reverse   # IMAP
 permit tcp any any eq 110 reverse   # POP3

ip access-list extended Citrix
 permit tcp any gt 1023 any eq 1494 reverse     # ICA (old)
 permit tcp any gt 1023 any eq 2598 reverse     # ICA (new)

# define an ordered group of services...

group Services HTTP Email Citrix

# define an interface matrix...

if-matrix Interfaces auto aliases=simple

# create a datafile of all interfaces for all services...

datafile rrd Apps Interfaces Services
```

Example #2 A datafile can also forgo matrices and simply combine multiple ACL groups. Here is the previous example reworked slightly:

```
ip access-list extended Site_A
 permit ip any Serial0/0 any exporter WANRTR        # serial0/0 is point-to-point

ip access-list extended Site_B
 permit ip any Serial0/1 65340 any exporter WANRTR  # serial0/1 goes to an MPLS provider with

ip access-list extended Site_C
 permit ip any Serial0/1 65341 any exporter WANRTR  # remote site BGP AS's 65340 & 65341

group Sites Site_A Site_B Site_C

# create a datafile of all sites for all services...

datafile rrd Apps Sites Services
```

Example #3 A datafile can specify more than two matrices/groups, though doing so is rare. Here is one situation where each remote site on an MPLS WAN has a unique AS. Each remote site exports NetFlow with the configuration "ip flow-export version 5 origin-as":

```
# define an interface matrix...

if-matrix Interfaces auto aliases=simple

# define an AS matrix...

as-matrix ASNs auto descriptions=asns.txt aliases=simple

# create a datafile of all AS's and all services...

datafile rrd Apps Interfaces ASNs Services
```

In the same directory as the flowage config file, there would be a file called asns.txt that names all the AS's that flowage will see:

```
Site_B   65340
Site_C   65341
# etc...
```

The web interface would let the user select Interfaces (e.g., "Site_B#Serial0/0"), AS's (e.g., the names "Site_B" and "Site_C" taken from asns.txt), and Services. This provides full visibility into mesh / site-to-site traffic usage. The downside is that since one rrd file is created for every service across each site-to-site mesh, thousands of rrd files may be created. If you do this, the **packed-rrd** datafile format may be required.

Example #4 It is also common to add an ACL to limit flows processed into a datafile. In the following example, the internet router is processed with one set of services and all other routers are processed with a second set of services.

```
exporter INET_RTR 10.10.10.10 public
exporter auto 10.0.0.0/8 public

# define an ACL that matches traffic from the internet router

ip access-list extended FROM_INET_RTR
 permit ip any any exporter INET_RTR

ip access-list extended NOT_FROM_INET_RTR
 deny ip any any exporter INET_RTR
 permit ip any any

# create different datafiles depending on the source of traffic...

datafile rrd InternetApps Interfaces InternetServices FROM_INET_RTR

datafile rrd EnterpriseApps Interfaces EnerpriseServices NOT_FROM_INET_RTR
```

Example #5 Another practical ACL helps speed up flow processing:

```
ip access-list extended Quickly
 permit ip any any packets gt 1

datafile rrd Apps Interfaces Services Quickly
```

The ACL "Quickly" only matches flows that have two or more packets. This trick filters out single-packet flows which are numerous on LAN-attached routers, and which rarely contribute to bandwidth usage but slow down flowage processing. This ACL also filters out security anomalies such as port scans and denial of service attacks.

SNMP INTERFACE CACHING

Each time flowage runs, it does a quick poll of every known exporter for its sysUpTime and ifTableLastChanged values. If these values indicate that the device has rebooted or its interface table has changed, or if more than 12 hours has passed, then a full poll is done.

SNMP cache files are text files stored in **directory cache** with names like ifData.**ip-address**[**device-name**]. The first four lines are:

sysUpTime	Last sysUpTime value seen.
ifTableLastChanged	Last ifTableLastChanged value seen.
lastPollTime	The time of last full poll. A full poll happens every 12 hours
sysName	The name of the device

The remaining lines are tab-delimited with four fields per line:

ifIndex	interface index number - this is the number referred to by flow records
ifAlias	interface name “GigabitEthernet 0/1”
ifDescr	interface description “Corporate LAN”
ifSpeed	interface speed, in bits per second

Flowage does a reasonable job maintaining these files for most environments. But there are situations where you may want to create and manage these files with a script of your own. In this case, run flowage.pl with the -nosnmp switch so that it doesn’t attempt any SNMP activity, but instead relies on the pre-existing cache files. A few reasons people have for managing SNMP themselves include:

- requirement for SNMPv3, which isn’t supported by flowage
- security restrictions that prevent the webview server from SNMP polling routers
- being able to manually override the interface name/description/speeds

DOMAIN NAME SYSTEM (DNS) USAGE

Reverse DNS lookups (in-addr.arpa) are a very useful tool for network administration, and flowage uses DNS in a number of ways.

Flowage.pl

Flowage only uses DNS to resolve IP addresses used in IP/nexthop matrix variables that have the **descriptions=dns** option. Although Flowage makes DNS queries, it does not use the responses in any way other than to make them available for other applications by storing them in a cache file.

Flowage's local DNS cache file ($tempDir/dns.txt) properly honors time-to-live (TTL), so only a small percentage of the IP addresses should need to be resolved at any given time. These are resolved using a "bulk" DNS query method that runs for 5 seconds and sends requests to the webview machine's default DNS servers. The 5 second timeout can be adjusted by the **dns timeout** parameter. Specific DNS servers can be set by the **dns servers** parameter.

The load of this query usually goes unnoticed on the DNS servers themselves. The volume of IP addresses is relatively small, since IP matrices should be limited to known subnets by using the **mask** option (e.g., **mask=192.168.10.0/24**).

Render.cgi

For speed, render.cgi uses the cache file stored by Flowage but does not do DNS lookups of its own. Thus, it is possible for the names to be stale.

Ad hoc query tool

The adhocFlow scripts have an option to perform DNS resolution on the results of ad hoc queries, using the same bulk DNS lookup function and local cache file as Flowage (configured in /etc/webview.conf). In practice, very large ad hoc reports may have incomplete DNS information. Repeating the report can often fill in the missing values.

Problems

If DNS is chronically not showing up in Render or adhocFlow, it could be that the DNS server or the network environment has problems with bulk DNS lookups. Windows NT 4.0 DNS, for instance, seems to either have real or artificial limits to the number of queries that can be serviced per second. In these cases, install a local DNS service on the machine where Flowage/adhocFlow runs and point flowage.cfg to use it. The local DNS service will perform its own caching function and won't have the limitations.

ACCESS CONTROL LISTS

Access Control Lists (ACLs) are written in a notation similar to Cisco IOS ACLs, which many people are familiar with. Each ACL is composed of one or more lines:

<permit|deny> <protocol> <source> <destination> [protocol options] [option]

Protocol

The protocol can be 'tcp', 'udp', 'icmp', 'gre', or a number 0-255. 'ip' matches all protocols.

Source and Destination

The source and destination of the flow each take the form:

[Interface] [AS] <IP/wildcard> [tcp/udp port range]

Interface

The interface can only be used if the ACL line also includes a matching **'exporter'** option (see below) and the config file also defines the 'exporter' (see 'Network Environment commands'). If these conditions are met, the ACL can contain statements such as 'permit ip Ethernet0/0 any Ethernet0/1 any'.

Autonomous System (AS)

The optional AS (Autonomous System number) is only useful if the NetFlow data export is being done from a router running BGP that has been configured with NetFlow V5 export. The AS is a number between 1 and 65535. There is not currently a way to match an AS range.

IP Address

The mandatory IP address can be **'any'**, **'host <ip>'**, **'<ip/bits>'** **'<ip> <wildcard>'**, or **'host @<name>'**. This last form references a host list defined statically by **'ip host-list'** or dynamically with **'dynamic'** (see the section of **'Data Manipulation Commands'**). Flowage follows the Cisco IOS convention of using a *wildcard* instead of a simple mask. E.g., a netmask of 255.255.255.0 would be a wildcard of 0.0.0.255. Wildcards have some nice attributes that can come in handy for people who speak binary. The rest of us use the ip/bits syntax.

Port range

The optional port range applies only to tcp and udp traffic. Ports can be numbers between 0 and 65535 or common protocol names (standard cisco port names and anything listed in /etc/services).

The syntax for specifying port ranges is:

eq <x>	*equal to x*	
gt <x>	*greater than x*	
ge <x>	*greater than or equal to x*	*(not cisco standard)*
lt <x>	*less than x*	
le <x>	*less than or equal to x*	*(not cisco standard)*
neq <x>	*not equal to x*	
range <x-y>	*greater than or equal to x and less than or equal to y*	

Protocol Options: TCP

Match flows that have one or more TCP flags:
fin, syn, rst, psh, ack, urg

To match flows without TCP flags, precede them with a '!'.

Positive flags are AND'd, but negative flags are OR'd. E.g., the sequence **"syn ack !rst !fin"** will match flows that have SYN *and* ACK, but don't have RST *or* FIN.

It's important to note that a flow's TCP flags are an amalgamation of all the packets in the flow. A flow could easily have SYN ACK FIN to all set. A flow with only a SYN indicates an unanswered connection request (many may be a port scan or SYN attack).

With IOS's **'ip flow active-timeout'** (11.x) or **'ip flow timeout active'** (12.0 and higher) configuration, a long TCP connection will be reported as several short flows, each with its own set of TCP flags.

Protocol Options: ICMP

Match flows with one or more ICMP messages. One can indicate the message by its ***type*** and ***code*** values (each between 0 and 255), but it's best to stick to standard Cisco ICMP keywords:

administratively-prohibited host-tos-redirect net-tos-unreachable redirect

alternate-address	host-tos-unreachable	net-unreachable	router-advertisement
conversion-error	host-unknown	network-unknown	router-solicitation
dod-host-prohibited	host-unreachable	no-room-for-option	source-quench
dod-net-prohibited	information-reply	option-missing	source-route-failed
echo	information-request	packet-too-big	time-exceeded
echo-reply	mask-reply	parameter-problem	timestamp-reply
general-parameter-problem	mask-request	port-unreachable	timestamp-request
host-isolated	mobile-redirect	precedence-unreachable	traceroute
host-precedence-unreachable	net-redirect	protocol-unreachable	ttl-exceeded
host-redirect	net-tos-redirect	reassembly-timeout	unreachable

Each ICMP packet shows up as a flow of its own, so a NetFlow analysis of ICMP messages is almost as good as an analysis from a sniffer.

Options

The following options are permitted for any protocols:

packets <range>

Match a count of packets in the flow.

bytes <range>

Match a count of bytes in the flow.

seconds <range>

Match a count of the elapsed seconds of the flow.

Bps <range>

Match the average **bytes** per second over the duration of the flow (bytes / seconds).

[kbps|mbps|gbps] <range>

Match the average **kilobits**, **megabits**, or **gigabits** per second over the duration of the flow (bytes * 125|125K|125M / seconds).

pps <range>

Match the average packets per second over the duration of the flow (packets / seconds).

packetsize <range>

Match the average packet size over the duration of the flow (bytes / packets).

next-hop [ip | ip/mask]

Match the next-hop of the flow. A tricky, but useful option.

exporter [ip | ip/mask]

Match the IP address or exporter name of the router that recorded the flow. This is required if source/destination interface names are also used. Only the names of explicitly defined exporters are allowed (i.e., dynamically learned exporters cannot be specified).

flow <%flow-list>

One or more parts of this ACL entry refer to a previously seen flow identified by a dynamic flow-list. See the **'dynamic'** statement for more information.

reverse

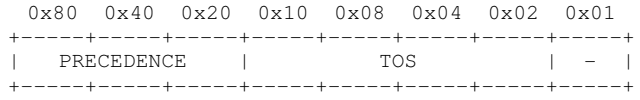
Create a duplicate rule with the source and destinations reversed. This is convenient if direction is irrelevant. E.g., 'permit tcp any any eq 23 reverse' matches telnet traffic in either direction.

tos <0-255>
dscp <ef|af11-af13|af21-af23|af31-af33|af41-af43|cs1-cs7|default|0-63>
precedence <routine|priority|immediate|flash|flash-override|critical|internet|network|0-7>
ecn <not-ect|ect|ect1|ect0|ce|0-3>

Match the value of the Type of Service (ToS) byte using one of four methods: **tos** matches the entire byte, **dscp** matches the upper 6-bits, **precedence** matches the upper 3-bits, and **ecn** matches the lower 2-bits. Common names, integer or hex values are all supported.

It's worthwhile noting that a Cisco flow is defined by protocol, IP(s), TCP/UDP ports, the input interface, ***and the ToS byte.*** Thus, otherwise identical packets with different ToS values are considered separate flows. This must be taken into account when Netflow-exporting routers fiddle with precedence/DSCP or implement Explicit Congestion Notification (ECN).

Below is a chart showing how the ToS byte is broken out according to DSCP, precedence, and/or ECN:

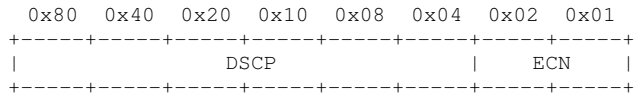


IP PRECEDENCE (RFC 791)

name	decimal	binary
network	7	111
internet	6	110
critical	5	101
flash-override	4	100
flash	3	011
immediate	2	010
priority	1	001
routine	0	000

IP TOS (RFC's 791/1349)

name	decimal	binary
min-delay	8	1000
max-throughput	4	0100
max-reliability	2	0010
min-monetary-cost	1	0001
normal	0	0000



DIFFERENTIATED SEVICES CODE POINT (DSCP) (RFC 2474)

name	decimal	binary
af11	10	001010
af12	12	001100
af13	14	001110
af21	18	010010
af22	20	010100
af23	22	010110
af31	26	011010
af32	28	011100
af33	30	011110
af41	34	100010
af42	36	100100
af43	38	100110

name	decimal	binary
cs1	8	001000
cs2	16	010000
cs3	24	011000
cs4	32	100000
cs5	40	101000
cs6	48	110000
cs7	56	111000
ef	46	101110
default	0	000000

EXPLICIT CONGESTION NOTIFICATION (ECN) (RFC 3168)

name	decimal	binary	description
not-ect	0	00	Non ECN-Capable Transport
ect(1)	1	01	ECN-Capable Transport
ect(0)	2	10	ECN-Capable Transport
ce	3	11	Congestion Experienced

EXAMPLES

EXAMPLE AUTHORIZATION FILE FOR AN INTERFACE MATRIX

```
REMOTE_ADDR=136.184.228.0/24          # for users on subnet 136.184.228.0/24
        wrtr-09                       #   access to any interface on "wrtr-09"
        wrtr-21#serial                #   access to any serial interface on "wrtr-21"
       fastethernet                   #   access to fastethernet interfaces on any router

REMOTE_ADDR=0.0.0.0/0                 # for everyone else
        fastethernet                  #   access to fastethernet interfaces on any router
```

EXAMPLE AUTHORIZATION FILE FOR A SUBNET MATRIX

```
REMOTE_NAME=bob
       10.48.0.0/16

REMOTE_NAME=sally
       10.32.0.0/14

REMOTE_NAME=admin
       0.0.0.0/0
```

EXAMPLE CONFIG FILE WITH GROUPS

```
---------start of config file---------

ip access-list ebxtended Mail
    permit tcp any any eq 25 reverse
    permit tcp any any eq 110 reverse
ip access-list extended FTP
    permit tcp any any eq 20 reverse
    permit tcp any any eq 21 reverse
group Services Mail FTP

ip access-list extended Closet01
    permit ip 10.0.1.0 0.0.0.255 any reverse
ip access-list extended Closet02
    permit ip 10.0.2.0 0.0.0.255 any reverse
ip access-list extended ServerFarm
    permit ip 10.0.3.0 0.0.0.255 any reverse
group Locations Closet01 Closet02 ServerFarm

datafile rrd Apps Locations Services

---------end of config file---------
```

This config file defines ACLs 'FTP' and 'Mail' and groups them together as 'Services'.

It also defines ACLs 'Closet_01', 'Closet_02', and 'Server_Farm' and groups them together as 'Locations'.

It's important to note that the because groups are exclusive, a flow from 10.0.1.0/24 to 10.0.3.0/24 would be tabulated as Closet01, not ServerFarm. This may or may not be what is expected and desired.

On disk, flowage will create 20 files that begin with Apps, one for every permutation of the group members and their 'Other' and 'Any' categories.

Apps.Closet01.Any.rrd	Apps.Closet01.FTP.rrd	Apps.Closet01.Mail.rrd	Apps.Closet01.Other.rrd
Apps.Closet02.Any.rrd	Apps.Closet02.FTP.rrd	Apps.Closet02.Mail.rrd	Apps.Closet02.Other.rrd
Apps.ServerFarm.Any.rrd	Apps.ServerFarm.FTP.rrd	Apps.ServerFarm.Mail.rrd	Apps.ServerFarm.Other.rrd
Apps.Other.Any.rrd	Apps.Other.FTP.rrd	Apps.Other.Mail.rrd	Apps.Other.Other.rrd
Apps.Any.Any.rrd	Apps.Any.FTP.rrd	Apps.Any.Mail.rrd	Apps.Any.Other.rrd

EXAMPLE CONFIG FILE WITH MATRICES AND GROUPS

```
---------start of config file---------
```

```
exporter Wan1 10.0.255.1 snmpABC
exporter Wan2 10.0.255.3 snmpABC

ip access-list extended Mail
    permit tcp any any eq 25 reverse
    permit tcp any any eq 110 reverse
ip access-list extended FTP
    permit tcp any any eq 20 reverse
    permit tcp any any eq 21 reverse
group Services Mail FTP

if-matrix Interfaces auto

datafile csv Flows Interfaces Services

---------end of config file---------
```

This config file defines ACLs 'FTP' and 'Mail' and groups them together as 'Services'.

It also defines a matrix called "Interfaces" that will match any router interfaces. To help out, two exporter commands tell flowage to load the interface tables from those routers using SNMP community string "snmpABC".

Assuming that the routers Wan1 and Wan2 each have Ethernet0 and Serial0 interfaces, we might find the following disk files created:

```
Flows.Wan1#Ethernet0.Mail.csv    Flows.Wan2#Ethernet0.Mail.csv
Flows.Wan1#Ethernet0.FTP.csv     Flows.Wan2#Ethernet0.FTP.csv
Flows.Wan1#Ethernet0.Other.csv   Flows.Wan2#Ethernet0.Other.csv
Flows.Wan1#Ethernet0.Any.csv     Flows.Wan2#Ethernet0.Any.csv
Flows.Wan1#Serial0.Mail.csv      Flows.Wan2#Serial0.Mail.csv
Flows.Wan1#Serial0.FTP.csv       Flows.Wan2#Serial0.FTP.csv
Flows.Wan1#Serial0.Other.csv     Flows.Wan2#Serial0.Other.csv
Flows.Wan1#Serial0.Any.csv       Flows.Wan2#Serial0.Any.csv
```

EXAMPLES OF DYNAMIC ACL'S (HEURISTIC MATCHING)

Below are examples of heuristic algorithms:

Identify known and likely BitTorrent traffic (host-list)

```
ip access-list extended Bittorrent_Signature
 permit tcp any ge 1024 any range 6881 6889

dynamic Bittorrent_Signature source-ip @Bittorrent_Clients timeout 120

ip access-list extended Bittorrent
 permit tcp any ge 1024 any range 6881 6889 reverse
 permit tcp host @Bittorrent_Clients ge 1024 any ge 1024 seconds gt 30 reverse
```

Identify Microsoft RPC traffic (flow-list)

```
ip access-list extended EPM
 permit tcp any ge 1024 any eq 135

dynamic EPM flow %MSRPC_Flows timeout 1

ip access-list extended MSRPC
 permit tcp any ge 1024 any eq 135 reverse
 permit tcp host $srcip ge 1024 host $dstip ge 1024 flow %MSRPC_Flows reverse
```

Fuzzy identification of Radio and VoIP traffic (flow-list)

```
! match long flows of small packets at a rate of 16-96 Kbps
ip access-list extended Audio_Senders
 permit tcp any any seconds ge 50 packetsize le 500 Bps range 2000 12000
 permit udp any any seconds ge 50 packetsize le 500 Bps range 2000 12000

dynamic Audio_Senders flow %Audio timeout 60

! "Radio" = flows where one side is a VoIP sender and the other isn't
ip access-list extended Radio
 deny   $protocol host $dstip eq $dstport host $srcip eq $srcport flow %Audio
 permit $protocol host $srcip eq $srcport host $dstip eq $dstport flow %Audio

! "VoIP" = flows where both sides are VoIP senders
ip access-list extended VoIP
 permit $protocol host $srcip eq $srcport host $dstip eq $dstport flow %Audio

! Be sure to list "Radio" above "VoIP" in your group of ACL's
```

Classify flows by where they were initiated (flow-list)

```
ip access-list extended LAN_Initiation
 deny $protocol $dstif host $dstip eq $dstport $srcif host $srcip eq $srcport
    flow %WAN_Sessions      ! skip flows already known in reverse
 permit ip Fast0/0 any Serial4/2 any exporter 172.30.0.17

ip access-list extended WAN_Initiation
 deny $protocol $dstif host $dstip eq $dstport $srcif host $srcip eq $srcport
    flow %LAN_Sessions      ! skip flows already known in reverse
 permit ip Serial4/2 any Fast0/0 any exporter 172.30.0.17

dynamic LAN_Initiation flow %LAN_Sessions timeout 90
dynamic WAN_Initiation flow %WAN_Sessions timeout 90

ip access-list extended LAN_Initiated_Sessions
 permit $protocol $srcif host $srcip eq $srcport $dstif host $dstip eq
    $dstport flow %LAN_Sessions reverse

ip access-list extended WAN_Initiated_Sessions
 permit $protocol $srcif host $srcip eq $srcport $dstif host $dstip eq
    $dstport flow %WAN_Sessions reverse
```