FlowScan: A Network Traffic Flow Reporting and Visualization Tool

Dave Plonka – University of Wisconsin-Madison

ABSTRACT

Internet traffic flow profiling has become a useful technique in the passive measurement and analysis field. The prerequisites for flow-based measurements are now available within the network infrastructure – particularly, in popular Cisco network devices. The integration of this feature has enabled the ''flow'' concept to become a valuable tool for the network administrator, as it had been in the past for the researcher.

This paper describes FlowScan, a software package for open systems that is freely available under the terms of the GNU General Public License. FlowScan analyzes and reports on flow data exported by Internet Protocol routers. It is an assemblage of perl scripts and modules and is the glue that binds together other freely available components such as a flow collection engine, a high performance database, and a visualization tool. Once assembled, the FlowScan system produces graph images, suitable for use in web pages. These provide a continuous, near real-time view of the network traffic through a network's border.

Although there are now a number of tools available that collect and process flow data, there is a dearth of visualization tools. By utilizing freely available software tools, FlowScan can be readily deployed in most modern educational institution, corporate, and ISP networks. The information presented by FlowScan assists in understanding the nature of the traffic that your network is carrying. It can be useful in the identification and investigation of anomalies such as poor performance and attacks on hosts. It can provide a foundation on which to develop usage-based billing or to verify the effectiveness of Quality-of-Service policies. By understanding the flows of traffic carried by the network, your institution should be able to make informed network management and bandwidth provisioning decisions.

Introduction

To better understand the nature of Internet traffic, the notion of *flow profiling* was introduced within the networking research community, and subsequently extended by other researchers, producing a series of useful findings. Flow profiling had been predicted to be relevant to applications such as route caching and usage-based accounting [ClaffyPB]. Today, based in part upon market demands for performance and accounting, flow profiling is built into networking devices. While not yet standards-based, the flow methodology is robust enough to persist through this period of vendor-specific implementations, and its benefits in many production networks warrant its early adoption.

Network administrators who collect measurement data often find that they either have collected too little data or too much of it. In a sense, flow profiling is a ''sweet spot'' between those extremes. Flows strike a balance between detail and summary. They are neither captured packets, nor are they merely aggregate totals tallied as packets travel across a given port or interface. Flows are an expressive abbreviation in which each flow represents a series of packets traveling between ''interesting'' end points. While flow features within the network infrastructure are a convenience, the presence of this feature alone is not sufficient for reliable continuous use in production networks. We need software tools to extract, record, and help us understand the flows.

A variety of tools for flow-based measurement have arisen from both the commercial and free software communities [NetFlow], FlowScan is one such freely available system. FlowScan is an assemblage of perl scripts and modules gluing together other freely available components, described below. Similar to other systems aimed at helping users to make sense of an overwhelming quantity of data, FlowScan means to simplify the *collection*, *storage*, and *visualization* of such data. Like most software tools, FlowScan has both ancestors and other relatives that played a part in defining its characteristics.

Several tools have set precedents for collection of network traffic data using passive measurement techniques. Of the freely available tools, MRTG [MRTG, Oetiker] and Cricket [Cricket, Allen] are among the most popular. Typically, these tools collect measurements by periodically collecting SNMP values, such as the interface or port counters named ifInOctets and ifOutUcastPkts, from routers and switches. They subsequently store this data in a reduced form that is easy to manage since it does not burden the user with database management responsibilities. These tools are ultimately effective because they make the data available to the end user in a useful, convenient manner. The design goals of FlowScan are similar to those of these tools.

FlowScan analyzes and reports on NetFlow data collected by CAIDA's cflowd [cflowd], a mature flow collection and analysis tool. This flow data is scavenged by FlowScan, which simply tries to discover interesting things about those flows, and maintains counters that reflect what is found. FlowScan then stores this myriad of counter values using Tobi Oeitker's RRDtool [RRDtool]. RRDtool is a a database system built from the ground up to effectively store and report on time-series data. Lastly, through the use of RRDtool and other front-ends, FlowScan provides reporting capabilities and visualization of the processed flow data.

FlowScan's Functionality

FlowScan and its component parts are responsible for collecting and processing raw flows exported from routers. FlowScan examines each flow and maintains counters based upon that flow's classification. FlowScan then periodically reports its results and may optionally take other actions. It may be configured to either archive or discard the raw flows after processing.

FlowScan, in its current form, supplies two report modules that illustrate its functionality: CampusIO and SubNetIO. CampusIO is a full-featured report module that is often the first and only report run by most FlowScan users. As such, its features are often thought to be those of FlowScan itself. The CampusIO report interrogates the raw flows, accumulates total counts and pushes these numeric statistics into high performance time-series Round Robin Databases [RRDtool]. Each database contains *packet*, *byte*, and *flow* counters. These counters are maintained in both in and out directions when appropriate.

Each Round Robin Database created by FlowScan contains between one and eight traffic statistics, stored at five-minute intervals, based upon one of these flow attributes:

- the IP protocol such as ICMP, TCP, and UDP
- the well-known service or application such as ftp-data, ftp, smtp, nntp, http, RealMedia, Quake, and Napster
- the class A, B, C network, or CIDR block in which a "local" IP address resides
- the AS (Autonomous System) pair between which the represented traffic was exchanged

Additionally, FlowScan maintains general traffic databases that contain total, multicast, and traffic involving unknown networks.

Figure 1 is a sample FlowScan graph of our campus traffic over a period of 24 hours, demonstrating some of the information handled by CampusIO.

This graph shows several features:

- There is a circadian rhythm to our campus traffic, with the low point centered about 6 AM and with peaks in the late evening.
- There is more total outbound traffic than inbound. That is, our campus consistently provides more Internet content than it consumes, regardless of whether or not our web cache is enabled.
- There is a significant amount of outbound FTP DATA content. This metric is a combination of both ftp-data and PASV mode ftp data transfers.
- Napster users were responsible for an amount of traffic both inbound and outbound that rivals or exceeds both general web traffic (HTTP) and file transfer traffic (FTP) [Plonka].
- Some sort of Napster outage occurred at approximately 3 PM local time.

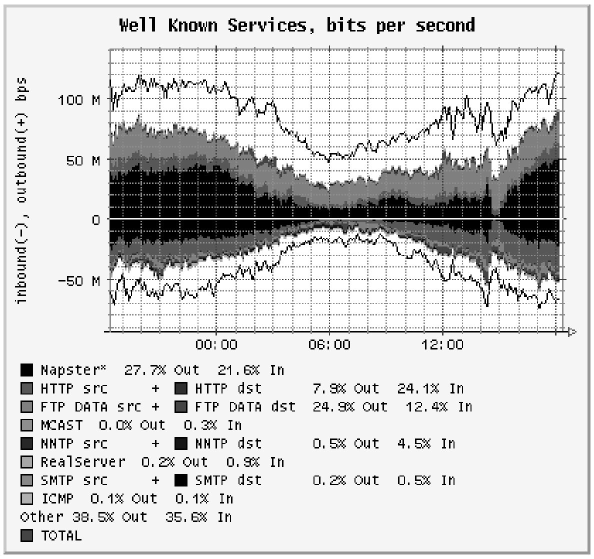

Figure 1: Graph of UW-Madison's campus traffic I/O during 24 hours, Sep 18 & 19, 2000.

The SubNetIO report requires a bit more configuration on the part of the installer, but it subsequently provides all the functionality of CampusIO, plus it maintains per-subnet statistics for applications such as billing a given campus "customer" based on their subnets' usage of precious bandwidth in and out of the campus itself.

Background on Flows and Cisco NetFlow

FlowScan utilizes flows defined and exported by Cisco's NetFlow feature. This flow definition is essentially that which was introduced in [ClaffyPB]. By this definition, an IP *flow* is a unidirectional series of IP packets of a given protocol, traveling between a source and destination, within a certain period of time. The source and destination are defined by IP address. Because the flow is unidirectional, nearly all useful exchanges between two hosts, such as a client and a server, are represented by at least two flows – one flow in each direction. For TCP and UDP flows, this definition considers the port number to be part of the source and destination address, making it convenient to determine which ''well known'' application such as HTTP and FTP is likely to have been responsible for the traffic represented by a flow. Additionally, Cisco NetFlow PDUs include the router's source and destination interface (named by the SNMP ''ifIndex'') and NetFlow version 5 PDUs include source and destination netmask and Autonomous System which are of interest to LAN and WAN engineers, respectively. FlowScan-1.003 requires Cisco NetFlow version 5 PDUs and makes use of all of these flow attributes.

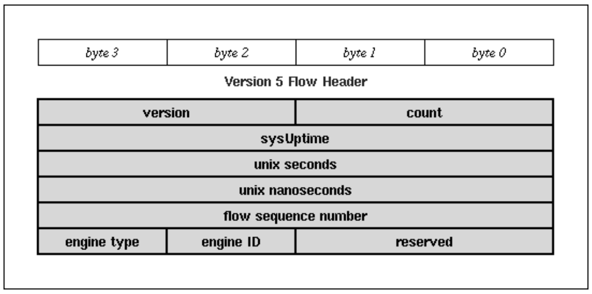

Figure 2: An image of Cisco's NetFlow V5 PDU header [McRobb].

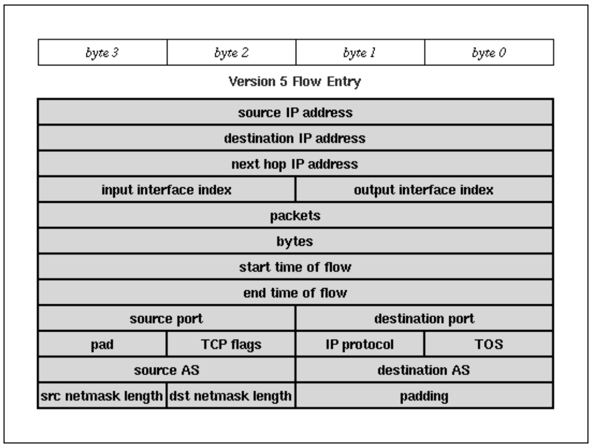

Figure 3: An image of Cisco's NetFlow V5 PDU entry [McRobb].

Note that while the term ''flow'' refers to the series of packets itself, NetFlow and FlowScan users often refer to the IP traffic flow accounting record that is exported from the router, and subsequently analyzed, as a ''flow''. This usage is natural because it is this exported accounting data that is the tangible data object that FlowScan manipulates. So, in this paper the term ''flow'' will most often have the latter meaning. Remember that this usage differs from the formal ''flow'' definition used in some other written works such as [ClaffyPB].

Cisco's NetFlow version 5 flow export format [Cisco]:

> Version 5 flow-export packets contain a flow header followed by a number of flow entries. The number of flow entries in the packet is in the count field in the flow header.
>
> Unlike version 1 flow-export, version 5 flow-export has AS numbers and netmask lengths for the source and destination. [McRobb]

While the NetFlow V5 PDU is well documented in [McRobb] and [Cisco], Figures 2 and 3 are here as a catalog of the flow attributes that are available to FlowScan. This set of attributes enables FlowScan's current capabilities, and to a degree ultimately limits them by imposing performance requirements and measurement compromises as described in the section on FlowScan problems.

FlowScan's Architecture

Hardware

A diagram of a basic FlowScan system's hardware components is show in Figure 4.

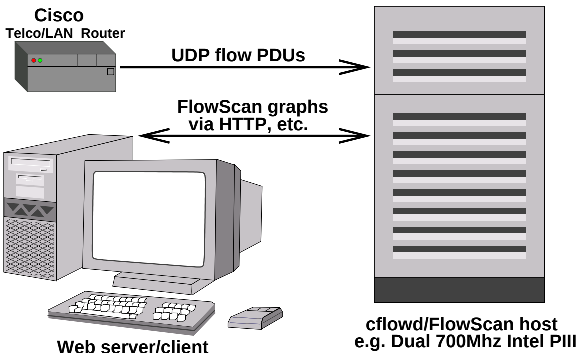

Figure 4: An image showing the basic hardware for the FlowScan System.

While FlowScan does not have strict platform requirements, most users who have successfully deployed it have dedicated either a SPARC machine running Solaris or Intel machine running GNU/Linux or BSD as a FlowScan system. Deploying on one of these platforms is also convenient because cflowd builds and runs on them as well, which allows one to co-locate FlowScan with cflowd so that both have access to the local disk. The fastest FlowScan machines appear to all be multi-processor Intel machines.

For further information, the installation documentation provided in the FlowScan distribution package discusses some recommended hardware parameters such as disk space and network interface card.

Software

A FlowScan system consists of a number of software components. The first such component is cflowd [cflowd], which is described thoroughly in [McRobb2]. FlowScan uses cflowd strictly as a flow collector. As such, the cflowd components used by FlowScan are the cflowdmux and cflowd programs. cflowdmux receives UDP Cisco version 5 flow PDUs from routers and and passes them to cflowd which writes them to disk in a portable, well-defined format of its own. FlowScan requires that the installer apply a patch to the cflowd sources. This modification enables cflowd to rotate and time-stamp its flow files at FlowScan's pre-defined sampling interval, which is typically five minutes. The decision to use five-minute samples was influenced by the popular tool MRTG [MRTG].

The second component is a program called flowscan; note that its name consists of only lower-case characters. "FlowScan" is the package whose primary procedural component is the program flowscan. This program is a perl script that is the central process in the system. It loads and executes report modules of the administrator's choosing. These report modules are simply perl modules that are derived from the FlowScan class defined in the FlowScan.pm perl module. FlowScan reporting modules are described below in the architecture section. As such, it is the flowscan script that actuates the whole system by maintaining databases of statistics regarding the IP traffic represented by the flows.

The third major component of FlowScan is RRDtool. RRDtool is described in [RRDtool] and is well documented in the supplied on-line manual pages. The FlowScan system uses RRDtool to store numerical time-series data and automatically distill or aggregate it into averages over time. Using RRDtool in this manner essentially replaces cflowd's arts++ data aggregation features, which have a different API, and no integrated graphing features such as those built into RRDtool. Specifically, RRDtool is used by FlowScan's supplied report modules to maintain a set of RRD files that form an extensive database of IP flow metrics. Also, RRDtool and RRGrapher are responsible for producing output such as graphs of IP traffic as GIF or PNG format image files.

The other components of the FlowScan package are utilities such as the make command, the Unix cron job-scheduling facility, and the gzip compression utility. Also, there is the flowdumper utility, supplied with the Cflow package, which is used to examine the raw flows "manually".

Figure 5 is a diagram of FlowScan's components and the data objects on which they operate.

FlowScan uses the disk as a large buffer area in which cflowd writes raw flow files that wait to be post-processed by flowscan. This buffering is an important fail-safe when used in networks with very high traffic or flood-based DoS attacks because FlowScan sometimes develops a backlog of flow files yet to be processed, which could total gigabytes in size.

Cisco's NetFlow was chosen as the base technology because of the ease of development within our campus' existing infrastructure.

Together with cflowd patches [patch], FlowScan enables multiple flow log analyses in one pass:

- flow archiving for post mortems
- real-time analysis with cflowd
- near real-time or post-processing with FlowScan

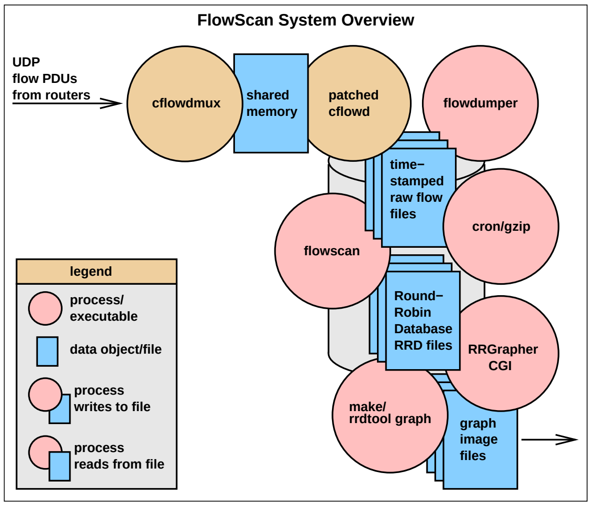

Figure 5: An overview of the software components of the FlowScan System.

Anatomy of a FlowScan Report Module

A FlowScan report module is an object-oriented perl module that has FlowScan as its base class. The module named FlowScan.pm implements the base reporting class, and serves as an example of what a FlowScan report must provide. Objects of a FlowScan-derived class have 3 methods, a.k.a. subroutines. These are:

perfile – Before flowscan processes a given time-stamped raw flow file produced by the patched cflowd, it invokes the perfile subroutine. As such, it is called once *per file*. The name of the file about to be processed is passed as an argument to this routine. For the convenience of derived classes, perfile also converts the time-stamp embedded within the raw flow file's name to a native Unix time_t representation. The names of the raw flow files produced by the patched cflowd are of the "flows.YYYYMMDD_HHMISS+TZ" format. For example, "flows.20000917_20:08:14-0500" is a file created at 8:08 PM on September 17, 2000 in a locale that is five hours west of GMT.

wanted – As flowscan reads each of the raw flows from within a cflowd-produced flow file, it invokes the wanted routine once per flow. This routine decides whether the current flow is *wanted*, i.e., whether it is an interesting flow for the report. The wanted subroutine is at the heart of a FlowScan report because it provides the opportunity for a report to interrogate the values stored in each flow and act accordingly.

report – After flowscan processes the last raw flow in a given flow file, it invokes the report routine. This routine is responsible for dispatching with any information "discovered" and collected by the aforementioned wanted routine. As such, this routine *report*s what has been discovered about the flows analyzed within the current flow file.

Two report classes are supplied with FlowScan-1.003: "CampusIO" and "SubNetIO". CampusIO is implemented in the file CampusIO.pm, following the usual naming convention for perl modules that implement object-oriented classes. CampusIO is derived from the FlowScan class defined in FlowScan.pm. SubNetIO is, likewise, implemented in SubNetIO.pm and its base-class is CampusIO. That is, SubNetIO is an extension that relies upon the functionality provided by CampusIO. As such, it is appropriate to run CampusIO *or* SubNetIO, but not both, since SubNetIO's functionality is a superset of the CampusIO's functionality.

Figure 6 is a simplified representation of the logic of CampusIO in perlish pseudo-code.

One of the features unique to the FlowScan system is its modular reporting structure. Because most of what the system does is the responsibility of peripheral modules, the flowscan script essentially just provides a framework for arbitrary testing and periodic reporting on flow content.

Stateful Inspection of Flows

The CampusIO report uses a number of heuristics that help it to identify elusive traffic, such as that of the Napster application [Plonka] or of PASV mode ftp file transfers. These heuristics employ a method of *stateful inspection*, which is essentially a variation of the stateful inspection of packets employed by many modern firewalls. Such firewalls track the state of an application session by observing information within a *packet* or series of *packets*. This technique enables the firewall to filter packets according to whether or not a session has been established and is still active. This state information is gleaned via passive inspection of either the packet header or application payload.

Similarly, FlowScan attempts to track the state of an application session, or series of sessions, by observing the information within *flows*. This flow-based stateful inspection enables counters to be maintained and reported upon for flows which would otherwise be left unidentified or be misidentified. For example, the identifying of traffic using a simple test of protocol and port number (i.e., tcp and port 6699) to identify Napster data flows is likely to sometimes erroneously match flows from applications such as PASV mode ftp which negotiate dynamic, unprivileged

```
package CampusIO;

sub perfile {
   # Remember the time-stamp from the filename.
   whence = filename2time_t(filename)
}

sub wanted {
   if (exporter_hop(flow::next_hop)) {
      # This flow is destined for another "local" flow-exporting router.
      # We'll catch it later, if and when it's exported by that router.
      return 0
   }

   if (outbound_hop(flow::next_hop) or outbound_interface(flow::output_if)) {
      # This flow is outgoing.
      outbound_total++
   } elsif (inbound_address(flow::destination_address)) {
      # This flow is incoming.
      inbound_total++
   } else {
      # This flow (an "intranet" flow) is unwanted.
      return 0
   }

   return 1
}

sub report {
   update_RRD_files(whence, inbound_total, outbound_total)
}
```

Figure 6: The logic of CampusIO.

port numbers. We minimize or eliminate this error by employing a more complicated test based on stateful inspection.

Figure 7 is a pseudo-code example of the logic used to detect Napster flows and PASV mode ftp data flows, which demonstrates the technique.

Also, without this sort of analysis, a large percentage of our campus traffic would be simply labeled as ‘‘unknown’’. As it is now, still more than 30% of our campus traffic remains unclassified.

FlowScan is not the only package that performs such stateful inspection of flows. Although FlowScan has for some time performed such inspection to identify Real Media flows and Napster flows, such a technique was developed independently by Simon Leinen for Fluxoscope [Fluxoscope] to identify PASV mode ftp data flows, a feature only more recently added to FlowScan.

Life with FlowScan

FlowScan produces a variety of graphs that provide a different view of network traffic than that provided by most other tools. For example, flow-per-second graphs are often as useful in network management as are packet-per-second and byte-per-second (bandwidth) graphs. FlowScan's RRD files are configured in such a way that they aggregate counters into less granular totals over time. For instance, five minute samples are combined into 30 minute samples, then into two hour samples, then into 24 hours samples. As a result, the sort of anomalies which we are able to discover is determined by the time length of the graph. In general we have found that by graphing over both

```
sub Napster_wanted {
   if (ICMP != flow::protocol || not (TCP == flow::protocol and
            (1024 < flow::srcport and 1024 < flow::dstport)) {
      return 0
   }
   # flow is either ICMP or TCP on unprivileged ports

   if (inbound(flow)) {
      direction = 'in';
      outside_addr = flow::srcaddr;
      inside_addr = flow::dstaddr
   } elsif (outbound(flow)) {
      direction = 'out';
      outside_addr = flow::dstaddr;
      inside_addr = flow::srcaddr
   }

   if (TCP == flow::protocol and ACK & flow::tcp_flags and
       is_napserver(outside_addr)) {
      # flow involves an ouside host that is a "publicly advertised" napserver
      remember_napster_server(outside_addr, flow::endtime);
      remember_napster_user(inside_addr, flow::endtime)
   } elsif (is_napuser(inside_addr)) {
      # flow involves an inside host that has talked to a napserver recently
      if ((TCP == flow::protcol and
           napster_ports(flow::srcport, flow::dstport)) or
          (ICMP == flow::protocol and 28 == flow::bytes/flow::pkts)) {
         # Confidence is high that this is a Napster application flow because
         # flow is either TCP on Napster "default" ports or
         # flow is ICMP using the "known" Napster ICMP packet size.
         napster_total++;
         return 1
      } else {
         # Confidence is lower that this is really a Napster application flow
         # because the port numbers are not Napster defaults or the ICMP
         # packet size isn't right.  We'll keep a count of these anyway.
         maybe_napster_total++
      }
   }

   return 0
}
```

Figure 7a: Pseudo-code logic to detect Napster flows and PASV mode ftp data flows, part 1.

short and *long* terms to be useful. However, each serves its own purposes.

Short-Term Analysis

By default, FlowScan's window for short-term analysis is 48 hours, that allows easy comparison of those two consecutive days. The standard graphs supplied with FlowScan provide views of traffic by network or subnet, by application or service, and by Autonomous System over the past 2 days. In each of these three graph categories, graphs are available by bits-per-second, packets-per-second, and flows-per-second.

```
sub ftp_PASV_wanted {
   if (TCP != flow::protocol) {
      return 0
   }
   # flow is TCP
   if (not (21 == flow::srcport or 21 == flow::dstport or
            (1024 <= flow::srcport and 1024 <= flow::dstport))) {
      return 0
   }
   # flow is either ftp (control port 21) or
   # TCP on unprivileged ports (>= 1024)
   if (1024 <= flow::srcport and 1024 <= flow::dstport) {
      # could be ftp PASV data flow
      if (ftp_session_exists(flow::srcaddr,
                             flow::dstaddr,
                             flow::endtime)) {
         # flow is probably ftp PASV data
         ftp_PASV_total++
      }
      return 1
   }
   # flow is ftp (control port 21)
   if (ACK & flow::tcp_flags) {
      # ftp (control port 21) stream is active
      remember_ftp_session(flow::srcaddr, flow::dstaddr, flow::endtime)
   }
   if (FIN & flow::tcp_flags) {
      # ftp (control port 21) stream is closed
      remember_ftp_session_closed(flow::srcaddr, flow::dstaddr, flow::endtime)
   }
   # this flow wasn't ftp PASV data
   return 0
}
sub perfile {
   # ...
   # Forget Napster server hosts (in the outside world) and user hosts
   # (on the inside) that haven't talked Napster recently.
   forget_quiet_napservers(30*60);
   forget_quiet_napusers(30*60);
   # Forget ftp session if we have seen FIN and 15 mins has passed
   # since we don't expect DATA flows so long after control port 21
   # has closed.
   # Also, forget ftp session if we haven't seen ACK on its control
   # port 21 in the past 30 mins since it's likely to have timed
   # out by now.  Perhaps we missed the flow which indicated that
   # the control port 21 stream has closed.
   forget_closed_or_quiet_ftp_sessions(15*60, 30*60);
   # ...
}
```

Figure 7b: Pseudo-code logic to detect Napster flows and PASV mode ftp data flows, part 2.

The graphs over a short, recent time frame are based upon the data that FlowScan keeps at five-minute intervals. The coarser-grained averaging in the longer term graphs will often hide anomalies seen in these finer-grained graphs. Network abuse, such as flood-based Denial of Service attacks, are easily visible.

Specifically, in our experience with FlowScan, we have learned that a discrepancy between the number of inbound and outbound flows or packets is an indication of abusive traffic, such as a DoS flood. Sudden changes in packet counts, especially when constrained to one protocol, are usually indications of a flood as well. Figure 8 is an example graph based on flow counts representing a flood of traffic which was unnoticeable on bandwidth usage graphs and nearly so on packet count graphs. The traffic responsible for the spikes in TCP flows was a flood of incoming 40-byte TCP ACK packets to which the campus host to which they were directed responded with 40-byte TCP RST packets. Even though a flood of small packets with dynamic source addresses produces spikes of equal magnitude in both packet and flow graphs, it is more readily noticed in the flow graph because the spike is a much larger proportion of the total flows than it is of the total packets.

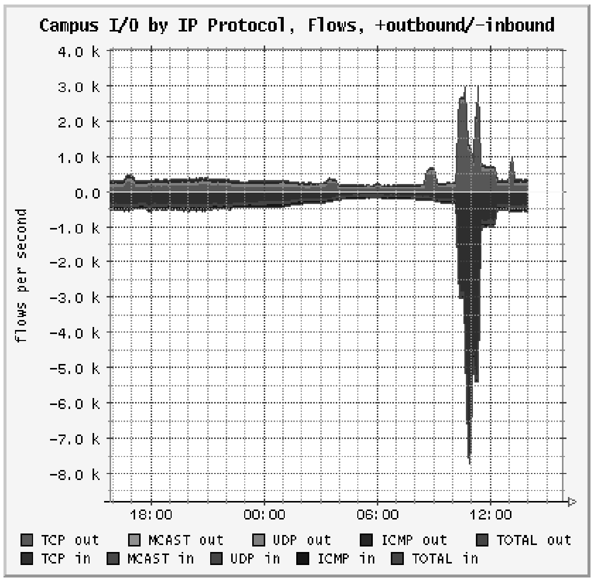

Figure 8: Graph produced 2000/09/24 showing a 40-byte TCP DoS flood.

When visualizing over the short term, it is important to remember that FlowScan increments traffic counters at the time *when the flows are exported*. More precisely, it records the values of these counters with the time-stamp corresponding to the five-minute interval in which cflowd wrote the flow to the raw flow file. As such, the time reported in the graphs is some function of the flow end time, but may involve timeouts defined in the NetFlow implementation. This time-stamp is thus not necessarily that when the represented traffic was actually observed by the router.

It is possible for a FlowScan graph to report a quantity of traffic in a given five-minute period that exceeds that actually forwarded by the router. On occasion, the quantity reported could even exceed the physical capacity of the link that carried it, which can be misleading. The reason that FlowScan operates in this way is twofold. First, for the sake of simplicity, FlowScan assumes that the traffic represented there-in must have occurred within the given five minute period. Without this assumption, we couldn't simply accumulate totals, and plot those values vs. time. Second, while NetFlow flows contain start and end time information for the packets in a given flow, they do not contain any hint as to how the delivery of those packets was distributed between the flow start and end time. So, even if FlowScan attempted to record the real time at which traffic was observed, it would not necessarily be any more accurate. We will see that the effect of this "time kludge" is negligible as data is coalesced into less granular time samples and visualized in longer-term analyses.

Long-Term Analysis

When producing FlowScan graphs, one may simply specify the number of *hours* over which to plot, such as 24 (hours) times 365 (days). Increasing the hours significantly beyond the default becomes an exercise in customization, since these graphs often need to be annotated with dates and other descriptions so that they have meaning to their intended audience.

Daily averages, which are used when graphing anything but fairly recent time ranges, hide spurious abuse activity such as flood-based Denial of Service attacks because such attacks are usually short-lived and will be severely minimized by averaging. As such the primary value of graphing over extended periods with daily averages is to aid in capacity planning and perhaps to help target traffic shaping efforts.

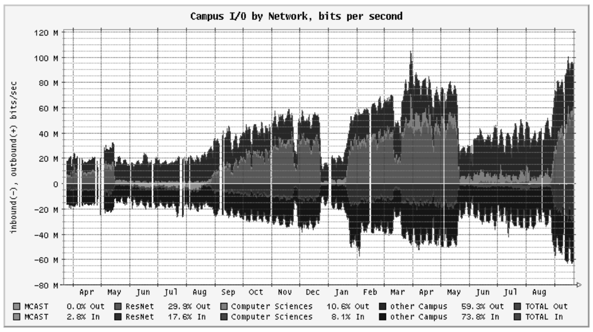

Figure 9: Graph produced 2000/09/21 showing campus traffic by network over the past 550 days.

Figure 9 shows a sample FlowScan long-term graph using daily averages over 550 days.

Information gleaned from this graph includes:

- The academic calendar dramatically influences the traffic levels, but only with respect to traffic to and from ResNet, the residence halls network.

- There has been an increase in outbound ftp traffic from the Computer Sciences department within the past year.
- While our outbound traffic level has consistently exceeded our inbound traffic level, the discrepancy between the two appears to be increasing.

Custom Graphs

In addition to using the "canned" graphs supplied with the FlowScan distribution, FlowScan users can produce custom graphs with the companion tool named RRGrapher [RRGrapher]. RRGrapher is the "Round Robin Graph Construction Set" and is implemented as a single perl CGI script, which runs under a web server that has access to RRD files. RRGrapher is also general front-end for RRDtool that allows users to interactively build graphs of their own design from any RRD files. Since it is based on RRDtool, not FlowScan, it allows one to generate graphs containing data from RRD files containing data from any RRDtool-based systems such as MRTG, Cricket, and FlowScan.

Figure 11 is a sample graph produced using RRGrapher. This graph shows the bandwidth used by ftp data transfers from ftp servers to ftp clients both in and out of the campus as determined by FlowScan. Figure 10 is the resulting RRDtool command that RRGrapher wrote and executed internally to produce

```
rrdtool graph /your/file/name/here.gif \
        --start 'Aug 15, 2000' \
        --end 'Sep 15, 2000' \
        --vertical-label 'bits/sec' \
        --title 'ftp PASV and ftp-data, +outbound/-inbound' \
        'DEF:A=ftp-data_src.rrd:in_bytes:AVERAGE' \
        'DEF:B=ftp-data_src.rrd:out_bytes:AVERAGE' \
        'DEF:C=ftpPASV_src.rrd:in_bytes:AVERAGE' \
        'DEF:D=ftpPASV_src.rrd:out_bytes:AVERAGE' \
        'CDEF:E=A,8,*,-1,*' \
        'CDEF:F=C,8,*,-1,*' \
        'CDEF:G=B,8,*' \
        'CDEF:H=D,8,*' \
        'COMMENT:A) ftp-data_src AVERAGE in_bytes' \
        'COMMENT:\n' \
        'COMMENT:B) ftp-data_src AVERAGE out_bytes' \
        'COMMENT:\n' \
        'COMMENT:C) ftpPASV_src AVERAGE in_bytes' \
        'COMMENT:\n' \
        'COMMENT:D) ftpPASV_src AVERAGE out_bytes' \
        'COMMENT:\n' \
        'AREA:E#0000FF:ftp-data src in (A,8,*,-1,*)' \
        'GPRINT:E:MIN:(min=%.1lf%S' \
        'GPRINT:E:AVERAGE:ave=%.1lf%S' \
        'GPRINT:E:MAX:max=%.1lf%S)' \
        'COMMENT:\n' \
        'STACK:F#00FFFF:ftp PASV src in (C,8,*,-1,*)' \
        'GPRINT:F:MIN:(min=%.1lf%S' \
        'GPRINT:F:AVERAGE:ave=%.1lf%S' \
        'GPRINT:F:MAX:max=%.1lf%S)' \
        'COMMENT:\n' \
        'AREA:G#00FF00:ftp-data src out (B,8,*)' \
        'GPRINT:G:MIN:(min=%.1lf%S' \
        'GPRINT:G:AVERAGE:ave=%.1lf%S' \
        'GPRINT:G:MAX:max=%.1lf%S)' \
        'COMMENT:\n' \
        'STACK:H#A0522D:ftp PASV src out (D,8,*)' \
        'GPRINT:H:MIN:(min=%.1lf%S' \
        'GPRINT:H:AVERAGE:ave=%.1lf%S' \
        'GPRINT:H:MAX:max=%.1lf%S)' \
        'COMMENT:\n'
```

Figure 10: RRDtool command created by RRGrapher.

the graph. At the user's option, RRGrapher displays this command when it produces the graph, allowing the user to cut and paste it elsewhere, perhaps to schedule it as a batch or cron job.

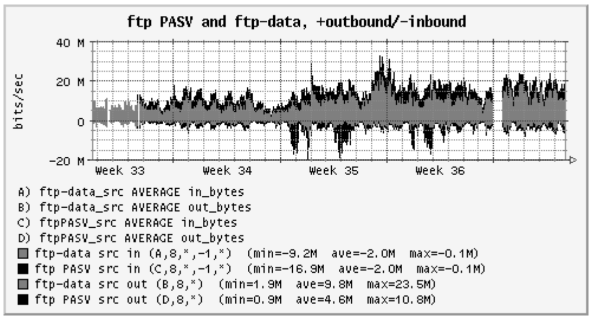

Figure 11: Graph showing campus ftp server traffic Aug 15, 2000 through Sep 15, 2000.

FlowScan Problems

FlowScan, while still extremely useful in most production networks, has exhibited a number of limitations and illuminated a number of problems. These may be instructive in the building of next generation flow analysis tools.

FlowScan's near real-time processing lags behind when processing flow files containing mostly "pathological" flows, such as those flows which represent only one packet per flow. These degenerate flows are produced to represent the flood of traffic produced during most Denial of Service attacks because the source or destination information is forged so that each packet contains a different IP address or port number than that which preceded it. In these situations the flow export rate approaches the router's packet forwarding rate, resulting in an avalanche of flows directed to the collection host. Figure 8 is an example graph of such traffic. Unless such traffic eventually ceases within hours or is blackholed by the network administrators, FlowScan would fall hopelessly behind, since it cannot process flows at rates approaching the packet forwarding rate of high performance routers with high capacity links.

Furthermore, under "normal" circumstances on some networks, FlowScan can still become buried in data. Even with the orders-of-magnitude reduction in the number of flows vs. the numbers of packets that those flows represent, flow processing tools may be overrun with too many flows to process in close to real-time on fast links. Currently, with Cisco's version 5 flow-export, FlowScan might not be able to scale beyond monitoring a couple fully-utilized OC3 (155 Mb/s) links. As technologies such as Gb ethernet and optical switching are deployed in border routers, the packet forwarding rate at these aggregation routers could outpace the performance gains in commodity personal computer hardware that are important for flow post-processing. Specifically, the performance adequacy of the processor and bus to persistent storage is in question. So, as packet forwarding performance improves, FlowScan-like processing may need to move from the network's DMZ to a location much closer to the end-user. This migration will require the deployment of arrays of measurement equipment. Managing an array of such devices will present new challenges for the network administrator.

In part, for tools other than FlowScan, these backlog issues have been avoided by aggregating totals on the router itself and then exporting those totals less frequently as summary PDUs. Another alternative is to adopt SNMP polling-based gathering of flow statistics rather than collecting unsolicited, exported flow data. However, both of these methods prevent nearly all the interesting post-processing that FlowScan currently performs, and also eliminate the archivable record of network traffic that detailed flow PDUs provide, which has proven invaluable during investigations of security compromises and network abuse.

Other problems occur when either NetFlow export or FlowScan is misconfigured. In complicated environments, configuring Cisco routers for cflowd has been easy to do incorrectly, especially since `ip route cache flow` should be enabled only on the appropriate interfaces. Misconfiguration can result in missing, skipped, or duplicated flows. Obviously, such misconfiguration may result in inaccurate flow statistics which, unfortunately, are dutifully stored into RRD files by FlowScan. These files are FlowScan's sole means of keeping archive data to answer historical questions or examine long-term trends. To combat this problem, in our use of FlowScan, we maintain a diary of interesting events for correlation with spurious graph anomalies.

Finally, identifying flows as representing traffic for specific applications is becoming increasingly difficult. Traditionally, the packets and flows of "well-known" applications were easily identified by interrogating values in the IP header such as port number. However, this identification is complicated by the increasingly popular technique of dynamically negotiating ports for connections, as well as the use of unreserved port numbers by many modern applications. Since flows do not contain the packet payload, we are not always able to label a given flow by application name with a high level of confidence, but instead must resort to compromises and heuristics. Furthermore, as VPNs, tunneling, and encryption become more common, network applications will intentionally hide the payload and even the IP header. Quality-of-Service initiatives may address some of these passive measurement problems since the same criteria by which packets will be marked for QoS purposes may be sufficient for flow identification as well. The identification of a flow's user and service class may have to suffice.

Future Directions and Possibilities

Alerts

Automatic event notification or *alert* capability would be a useful addition to FlowScan. A report module named "CampusIOAlert" is currently under development which dispatches alerts to email addresses or alphanumeric pagers. We have focused on alerts based on tests that would be unlikely to generate false positives and would not need to be executed more than once per flow file. For instance, we have implemented an alert to report flood-based Denial of Service attacks based simply on the detection of an imbalance of inbound versus outbound flows.

The opportunity for alert modules to perform tests on individual flows may also be useful, although more processing intensive since such tests must be executed once per flow rather than once per flow file. For example, we have had good luck in identifying hosts that were infected with a specific email-attachment worm because this worm attempted to negotiate an HTTP connection from the infected host to a well-known destination host.

Ultimately, with a report such as CampusIOAlert, FlowScan users will be able to write their own alert tests using the expressive syntax of the perl language itself. These alerts will also be able to test values from RRDtool's `rrdtool graph` command, which has expressive RPN expression evaluation and test features of its own.

Performance Enhancements

One of the largest challenges has been dealing with flow file backlog created whenever the quantity of flows exported by the router, and subsequently written to disk, exceeds the quantity that FlowScan is able to process in near real-time.

Converting FlowScan into a multi-threaded, or simply multi-process, application was thought to be a viable solution to the backlog issue, and was recently implemented by Alexander Kunz, an ambitious FlowScan user. This multi-process flowscan enlists the help of another flowscan process whenever FlowScan had more than one flow file yet to be processed. A mutex technique in which each process takes a numbered "ticket" was employed to ensure that subsequent updates of the RRD files occurred in the proper time-series order.

At the time of this writing, September 2000, the challenge of integrating this multi-process patch into the mainline FlowScan distribution has been delayed in favor of simply improving the performance of the single-threaded model. A faster single-threaded solution avoids those complications inherent in multi-threading which we have yet to address, namely that the FlowScan code would need to enforce serialized access to its internal data structures used for stateful inspection. Also, light-weight thread support in perl, while available, is still considered experimental.

Instead, to vastly improve FlowScan's single-process performance, IP address prefix matching was reimplemented with Patricia Trie lookups in a C language extension to perl – i.e., a perl module. The term "Trie" is derived from the word "retrieval" but is pronounced like "try". Patricia stands for "Practical Algorithm to Retrieve Information Coded as Alphanumeric" and is thoroughly described in [WrightS]. The Patricia Trie performance characteristics are well-known as it has been employed for routing table lookups within the BSD kernel since the 4.3 Reno release [Sklower]. We chose the Patricia data structure and search algorithm upon realizing that FlowScan's decisions based upon IP addresses are nearly identical to those decisions that an IP router must make when delivering a packet based upon destination IP address.

Multi-vendor Compatibility

While nearly all the focus so far has been on Cisco's NetFlow feature, other network hardware vendors have developed similar flow-based accounting technology. Juniper, with its announcement of its JUNOS software release 4.1R1 in August 2000, now supports a sub-set of the accounting functionality using the Cisco NetFlow-defined PDU formats. Riverstone Networks, formally a portion of Cabletron, has a protocol called LFAP: Lightweight Flow Accounting Protocol. Related software, such as `slate`, is available at [NMOps].

We are looking at the possibility of supporting flow processing based on the implementations of these or other vendors. Potentially one could build this support for FlowScan by enhancing cflowd to support other flow export implementations, by modifying other collectors to produce the cflowd-defined raw flow file format, or by modifying FlowScan or the underlying Cflow perl module to be able to process a stream of flows of a different format.

Availability

FlowScan is freely available under the terms of the GNU General Public License [FlowScan, GPL]. FlowScan-1.002, that was released March 21, 2000, was used at approximately 100 sites. FlowScan-1.003 was released September 15, 2000. More information on FlowScan is available at http://net.doit.wisc.edu/˜plonka/FlowScan/ . A mailing list for flowscan users is archived at http://net.doit.wisc.edu/˜plonka/list/flowscan/ .

Likewise, RRGrapher is freely available under the same terms at http://net.doit.wisc.edu/˜plonka/RRGrapher/ .

FlowScan's primary building blocks, cflowd and RRDtool were both developed with the support of CAIDA. They are are freely available at http://www.caida.org/tools/measurement/cflowd/ and http://ee-staff.ethz.ch/˜oetiker/webtools/rrdtool/ , respectively.

Summary

FlowScan has become a useful tool for our network engineering team. We have run it nearly continuously to measure and analyze all traffic to and from our campus since late 1998. The resulting reports and graphs have been of use for the detection of abuse and other anomalies that are detrimental to the network backbone's performance as a whole. Without such a tool, these incidents would have gone unexplained. Also, FlowScan's long-term graphs have been used to convey the impact of Napster to our management, and to justify progressive upgrade and bandwidth acquisition plans for the campus. With its public release, FlowScan has proven useful for these and other purposes by a user base of educational institutions, government institutions, corporations, and Internet service providers.

When attempting to meet the challenges of managing a heavily utilized IP network, near real-time traffic analysis and visualization quickly becomes an essential technology. One way to provide these capabilities is by utilizing Internet traffic flow profiling based on technology available in most networking equipment. FlowScan is a system designed to provide this analysis continuously in near-real time and can be an effective tool to better understand Internet traffic.

Acknowledgements

The initial public FlowScan, released in September 22, 1999, was patiently tested by a number of volunteers. The effort, feedback, and encouragement of those users and of members of the flowscan mailing list, has been much appreciated and has made FlowScan a better tool. James Deaton of OneNet, Ted Frohling of the University of Arizona, John Kristoff of DePaul University, Gregory Goddard of the University of Florida, and others have made either a private or public FlowScan-based web site accessible to myself and other FlowScan users. This information has been invaluable for research and development. Alexander Kunz of Nextra in Germany and Michael Hare of the University of Wisconsin-Madison have made tangible contributions to FlowScan. Alexander hacked out the first multi-threaded FlowScan, prototyped a visualization enhancement for the graphs, and made the changes available to the user community. Michael enthusiastically began development of the aforementioned CampusIOAlert report module. Daniel McRobb, Tobi Oetiker, and CAIDA have provided the main tools upon which FlowScan is built, namely cflowd and RRDtool. Thanks to K. Claffy, of the Cooperative Association for Internet Data Analysis (CAIDA), and Denis DeLaRoca, of the University of California, Los Angeles who have helped by providing their thoughts and encouragement on this and related work.

Author Information

Dave Plonka has developed a number of free software packages, many of which are network management tools. He works as a systems programmer doing network engineering within the Division of Information Technology (DoIT) at the University of Wisconsin – Madison. In the formative years of his working life he was a programmer in the commercial software industry focusing on the development of portable libraries and relational database applications under VMS and Unix. In 1991, he received a B.S. in Computer Science from Carroll College in Waukesha, Wisconsin. Dave can be reached at plonka@doit.wisc.edu or via http://net.doit.wisc.edu/~plonka/ .

References

[ClaffyPB] K. Claffy, G. C. Polyzos, and H.-W. Braun, ''Internet traffic flow profiling'', UCSD TR-CS93-328, SDSC GA-A21526, http://www.caida.org/outreach/papers/itf.html , November 1993.

[NetFlow] Simon Leinen, ''FloMA: Pointers and Software, NetFlow'', http://www.switch.ch/tf-tant/floma/software.html#netflow .

[MRTG] "MRTG", http://ee-staff.ethz.ch/~oetiker/webtools/mrtg/ .

[Oetiker] Tobias Oetiker, ''MRTG – The Multi Router Traffic Grapher'', USENIX LISA '98 Conference Proceedings, 1998.

[Cricket] ''Cricket Home'', http://cricket.sourceforge.net/ .

[Allen] Jeff R. Allen, ''Driving by the Rear-View Mirror: Managing a Network with Cricket'', USENIX NETA '99 Conferences Proceedings, 1999.

[RRDtool] ''RRDtool – Round Robin Database Tool'', http://ee-staff.ethz.ch/~oetiker/webtools/rrdtool/ .

[Plonka] Dave Plonka, ''UW-Madison Napster Traffic Measurement'', 2000., http://net.doit.wisc.edu/data/Napster/ .

[Cisco] ''Cisco's IOS NetFlow feature'', http://www.cisco.com/warp/public/732/netflow/ , http://www.isco.com/warp/public/cc/cisco/mkt/ios/netflow/tech/napps_wp.htm .

[McRobb] Daniel W. McRobb, ''cflowd configuration'', http://www.caida.org/tools/measurement/cflowd/configuration/configuration.html , 1998-1999.

[cflowd] ''cflowd – CAIDA's flow analysis tool'', http://www.caida.org/tools/measurement/cflowd/ .

[McRobb2] Daniel W. McRobb, ''cflowd design'', http://www.caida.org/tools/measurement/cflowd/configuration/design/design.html , 1998.

[patch] ''patches to cflowd'', http://net.doit.wisc.edu/~plonka/cflowd/ .

[Fluxoscope] Simon Leinen's ''Fluxoscope'', http://www.switch.ch/lan/stat/fluxoscope/ .

[Leinen] Simon Leinen, ''Fluxoscope: a System for Flow-based Accounting'', 2000., http://www.tik.ee.ethz.ch/~cati/deliv/CATI-SWI-IM-P-000-0.4.pdf .

[RRGrapher] Dave Plonka, ''RRGrapher – the Round Rober Grapher, a Graph Construction Set for

RRDtool'', http://net.doit.wisc.edu/~plonka/RRGrapher/ .

[WrightS] Gary R. Wright, W. Richard Stevens, *TCP/IP Illustrated, Volume 2: The Implementation*, Addison-Wesley Publishing Company, Inc., 1995.

[Sklower] Keith Sklower, ''A Tree-Based Packet Routing Table for Berkeley UNIX'', *Proc. USENIX Conference Proceedings*, http://www.cs.berkeley.edu/~sklower/routing.ps , 1991.

[NMOps] ''Network Management Operations'', http://www.nmops.org/ .

[FlowScan] "FlowScan", http://net.doit.wisc.edu/~plonka/FlowScan/ .

[GPL] ''GNU General Public License'' Version 2, Free Software Foundation, Inc., http://www.gnu.org/copyleft/gpl.html , 1991.