Flow Analysis Tool Whitepaper

Nichole Boscia
NASA Advanced Supercomputing Division
NASA Ames Research Center
Moffett Field, CA 94035-1000
Nichole.K.Boscia@nasa.gov

I. Introduction

Faster networks are continually being built to accommodate larger data transfers. While it is intuitive to think that implementing faster networks will result in higher throughput rates, this is often not the case. There are many elements involved in data transfer, many of which are beyond the scope of the network itself. Although networks may get bigger and support faster technologies, the presence of other legacy components, such as older application software or kernel parameters, can often cause bottlenecks.

Engineers must be able to identify when data flows are reaching a bottleneck that is not imposed by the network and then troubleshoot it using the tools available to them. The current best practice is to collect as much information as possible on the network traffic flows so that analysis is quick and easy. Unfortunately, no single method of collecting this information can sufficiently capture the whole end-to-end picture. This becomes even more of a hurdle when large, multi-user systems are involved. In order to capture all the necessary information, multiple data sources are required.

This paper presents a method for developing a flow analysis tool to effectively collect network flow data from multiple sources and provide that information to engineers in a clear, concise way for analysis.

The purpose of this method is to collect enough information to quickly (and automatically) identify poorly performing flows along with the cause of the problem. The method involves the development of a set of database tables that can be populated with flow data from multiple sources, along with an easy-to-use, web-based front-end interface to help network engineers access, organize, analyze, and manage all the information.

Network Flow Data Sources

Network engineers rely on flow data from their network routers, typically using industry standard protocols like NetFlow or sFlow. However, this information is limited to the source and destination Internet Protocol (IP) addresses, service port, and amount of data. On multi-user systems, there is no easy way to map this data to a specific user. More importantly, it does not capture critical flow characteristics such as packet loss (as retransmits), TCP window sizes, or TCP protocol options—all of which are crucial to troubleshooting performance issues.

Network taps and their respective collectors do a much better job of obtaining detailed TCP session parameters to aid in flow analysis. When placed in an area that will see all traffic in and out of the network, they are key for collecting information such as the number of retransmits, window sizes, and even jitter. However, like the flow sampling done from the routers, network taps still have the limitation of not being able to map the flow to a user or provide any information from whichever

application the user is running, such as which file is being transferred. Collectors or sensors must perform additional processing to aggregate the flow data, which is often dumped every minute. For large-scale networks, this requires significant storage space and specialized formatting, making it impossible to store in a database for simple queries and requiring proprietary software to parse through the data.

The missing layer in the above techniques is data from the user's file transfer application itself (such as BBFTP, SCP, BBCP, GridFTP, etc.). That information can provide important details about who is transferring the data, where the data is being read from and written to, and what parameters are being defined by the application. It is important to note the source of the data being transferred because it can sometimes come from a mounted, remote source that becomes a bottleneck. Additionally, if the optional parameters of the application are logged, they can be used to determine how many data streams the user has defined. With some applications, the data ports are randomized and it is difficult to identify which ports are bound to the same parallel transfer. In this case, data is collected and reported by the server application itself, which often requires some software customization, so information can vary.

One other popular method of collecting flow information is applying the custom kernel patch for Web100 (or Web10G) and using the Userland API to collect data about each flow. Similarly to network taps, this method can collect and record very specific information about each TCP session. In situations where taps cannot capture all the network traffic, or where there are a limited number of systems used for file transfers, the Web100 kernel can provide an alternative source of information. However, this approach requires high upfront overhead to write customized code and integrate it into the applications or system to record the information. The upside is that this option will give the most information about each session.

Table 1. Common network data collection options and their limitations.

	User Name	**Transfer Rate**	**Packet Loss**	**Window Sizes**	**Latency**	**Application Options**	**TCP Options**
NetFlow/sFlow	No	Estimated	No	No	No	No	No
Network Taps	No	Estimated	Yes	Yes	Yes (Jitter)	No	Yes
Application	Yes	Yes	No	No	No	Yes	No
Kernel (Web100)	No	Yes	Yes	Yes	Yes	No	Yes

None of these methods can individually provide all the data that is needed for complete flow analysis. In order to capture a complete picture for analysis by the engineers, information must be collected from multiple sources and correlated. This paper outlines methods for accomplishing this in a way that is cost-effective, robust, and intuitive for engineers to use. The resulting flow analysis tool goes beyond just collecting flow data—it also includes a management front end that incorporates a ticket-like system, as well as additional features such as visualization. This paper will outline a full-featured network flow data collection, monitoring, and analysis system. For a minimalistic approach, sections can be skipped that are not of interest to the reader.

Implementation Methods

All methods discussed in this paper are performed using open source software. Additional information on where to download the software can be found in the appendix. Depending on the data collection options selected from Table 1, not all of the components presented may be needed. Figure 1 below shows an overview of a data collection and server set-up for a possible implementation of the tool presented here.

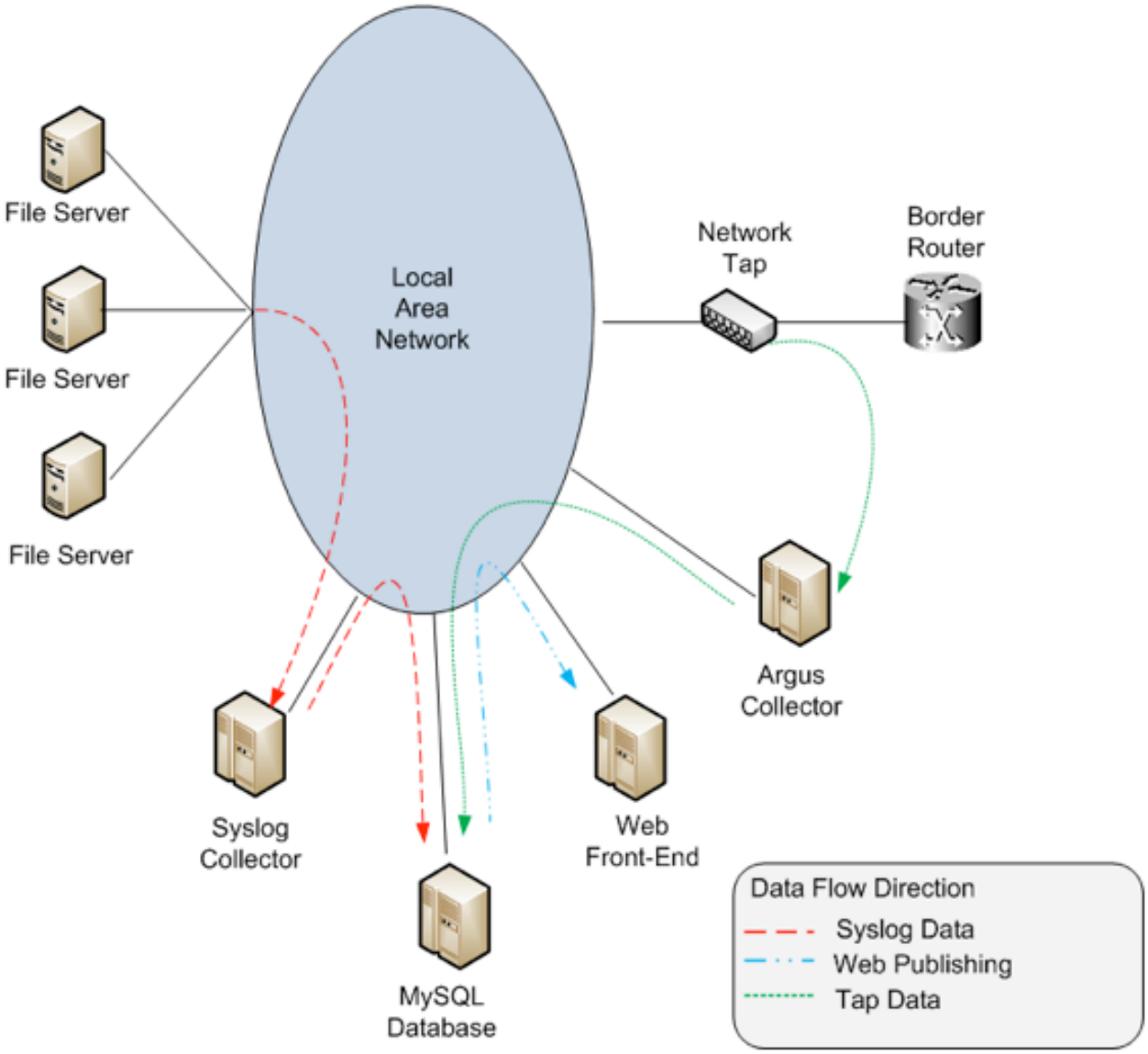


Figure 1. Diagram of a possible data collection scenario and how data would flow between servers.

While this paper uses specific programming languages in its examples, it is up to the reader to determine which language they prefer to use. The languages and tools used in this paper are software packages available for download on the Internet, and are considered standard use in most network environments. For implementing network tap data collection, Argus is the method-of-choice because it offers a distributed system environment with an aggregated network point.

The languages and tools used for the various components of the implementation presented here are:

- Perl: back-end data collector
- PHP: front-end interface for web application
- MySQL: database back-end
- Apache: web server
- Argus, Flowd, or Web100: collection of detailed session information

II. Setting up a Database

This tool uses several database tables, which are divided by function. The tables are: flowData, flowDetails, flowMapping, status, flowNotes, and flowHistory. Some of these tables are optional but will still be mentioned here, and several additional table options will be discussed in Section V.

The flowData and flowDetails tables contain the bulk of the data and are both required for minimal flow analysis. The other tables (flowMapping, status, flowNotes, and flowHistory) are for administrative and management purposes but are essential if the front-end web interface is going to be used for a large number of flows that require tracking.

The following subsections present the setup of these database tables. Details on how to obtain data and populate the tables will be discussed in Section III. The actual syntax for creating the tables is included in the appendix. Figure 2 below shows a diagram of the database tables, and how data would flow between them.

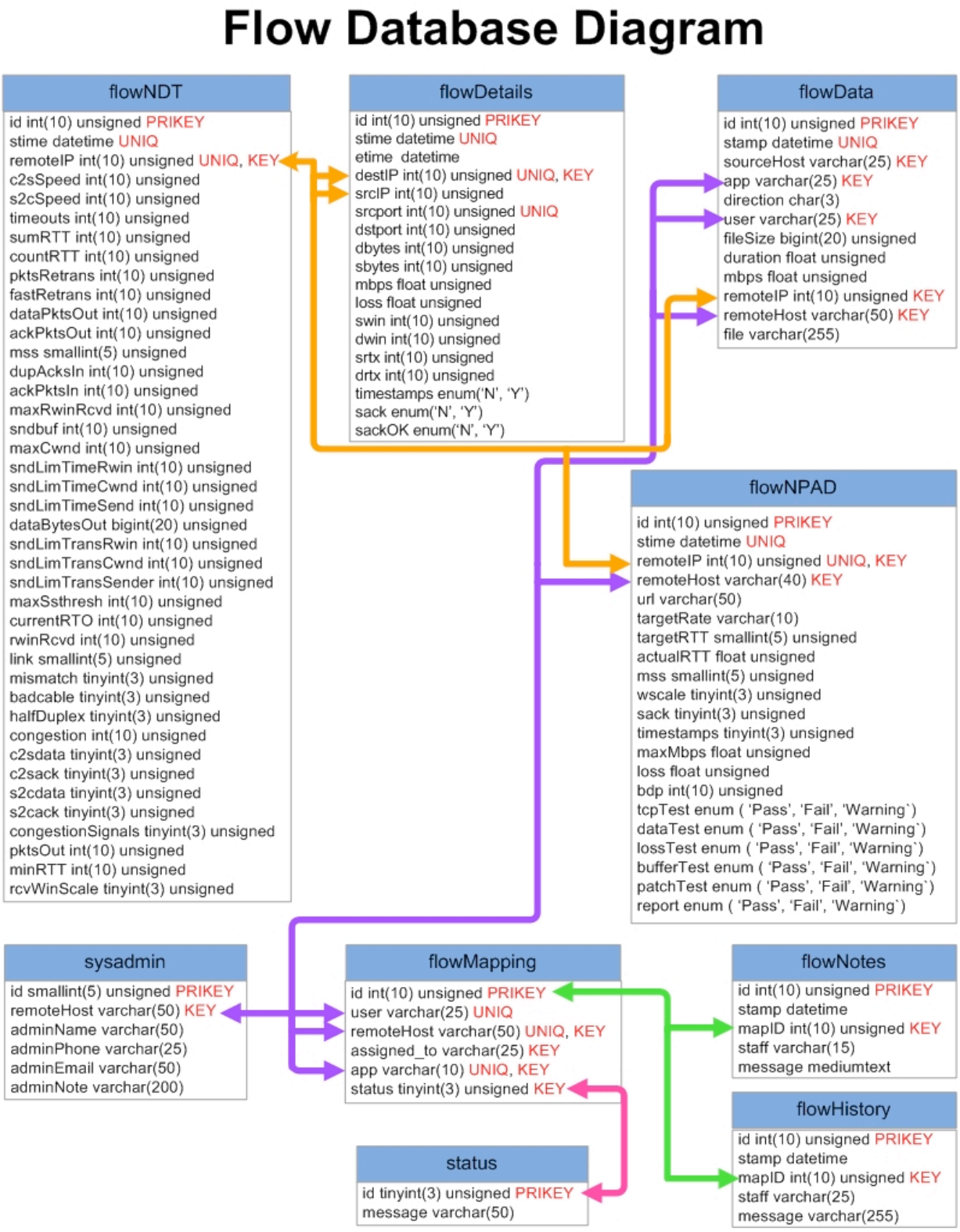


Figure 2. Relational flow diagram of database tables and indices.

The flowData Table

The flowData table is the core of the tool since it stores the general flow characteristics that are collected by the user's application. When a user runs the software, it creates a log entry in the system's syslog file. This log file is then parsed by a back-end script, which inserts the fields into the database. The script will vary depending on how the application logs the data, and not all fields may be captured.

Table 2. The flowData table setup.

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_increment
stamp	datetime	YES	MUL	NULL	
sourceHost	varchar(25)	YES	MUL	NULL	
app	varchar(25)	YES	MUL	NULL	
direction	char(3)	YES		NULL	
user	varchar(25)	YES	MUL	NULL	
fileSize	bigint(20) unsigned	YES		NULL	
duration	float unsigned	YES		NULL	
mbps	float unsigned	YES		NULL	
remoteIP	int(10) unsigned	YES	MUL	NULL	
remoteHost	varchar(50)	YES	MUL	NULL	
file	varchar(255)	YES		NULL	

flowData field descriptions:

- *id* – a unique ID for each entry
- *stamp* – a timestamp of when the transfer was logged
- *sourceHost* – the host the information was collected from (note: this does not necessarily mean it initiated the transfer)
- *app* – the file transfer application that was used (this may also be enumerated)
- *direction* – an application-based direction, such as a PUT or a GET
- *user* – the username of the person who initiated the transfer
- *fileSize* – the size of the file in bytes
- *duration* – how long the transfer took
- *mbps* – the transfer rate achieved, in megabits per second, as calculated by the application
- *remoteIP* – the IP address of the remote host
- *remoteHost* – the hostname of the remote host at the time of the transfer
- *file* – the path and file being transferred

Note that the sourceHost IP address isn't recorded, just the hostname. Both can be captured if the application logs which local IP address is used. In cases where a system has many network interfaces, logging multiple IP addresses can make running queries more complex. For simplicity, just the hostname is logged, since that provides enough information for the engineers.

The flowDetails Table

The information pulled from the network taps (or a Web100 kernel) goes into a separate flowDetails table. It is nearly impossible to map the flows between the application data from the flowData table and the data collected from the network taps because of variations in timestamps and aggregation of data done by the file transfer application. The flowDetails table is essential for troubleshooting because it

includes information that can't be obtained from other methods. Additionally, Argus can provide more details than those displayed here and more fields can easily be added to the table as needed. In the near future, TCP options such as winscale will also be retrievable from the Argus flow records. If the Web100 kernel is used, some additional important values, such as the round-trip time, max pipe size, fast retransmits, and max MSS can all be captured as well.

Table 3. The flowDetails table setup.

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_increment
stime	datetime	YES	MUL	NULL	
etime	datetime	YES		NULL	
destIP	int(10) unsigned	YES	MUL	NULL	
srcIP	int(10) unsigned	YES		NULL	
srcport	int(10) unsigned	YES		NULL	
dstport	int(10) unsigned	YES		NULL	
dbytes	int(10) unsigned	YES		NULL	
sbytes	int(10) unsigned	YES		NULL	
mbps	float unsigned	YES		NULL	
loss	float unsigned	YES		NULL	
swin	int(10) unsigned	YES		NULL	
dwin	int(10) unsigned	YES		NULL	
srtx	int(10) unsigned	YES		NULL	
drtx	int(10) unsigned	YES		NULL	
timestamps	enum('N','Y')	YES		NULL	
sack	enum('N','Y')	YES		NULL	
sackOK	enum('N','Y')	YES		NULL	

flowDetails field descriptions:

- *id* – unique ID for each entry
- *stime* – start time of the flow
- *etime* – end time of the flow
- *destIP* – IP address of the destination system (note: this is directional-specific)
- *srcIP* – IP address of the source system (note: again, directional-specific)
- *dstport* – port number used by the destination system (protocol type is assumed to be TCP and isn't explicitly captured since file transfer software is usually TCP)
- *srcport* – port number used by the source or initiating system
- *dbytes* – number of bytes transferred to the destination
- *sbytes* – number of bytes transferred to the source
- *mbps* – throughput rate achieved by the flow, as calculated by Argus (in megabits per second)
- *loss* – total percentage of packets lost during the transfer
- *swin* – maximum size of the TCP window used by the source system (in bytes)
- *dwin* – maximum size of the TCP window used by the destination system (in bytes)
- *srtx* – number of retransmitted packets from the source
- *drtx* – number of retransmitted packets from the destination
- *timestamps* – whether or not the TCP timestamps option is used
- *sack* – whether the initiating host wants to use selective ACK
- *sackOK* – defined if the session negotiates to use selective ACK

The flowMapping Table

The flowMapping meta table simply creates entries with unique groupings that are used to easily identify a particular flow analysis case. A case is a method for tracking the status of flows being worked on by engineers—similar to a basic helpdesk ticketing system. The key is determining which set of identifiers constitutes a unique case. For our purposes, the unique combination is a ternary set of user:remotehost:application. The bottleneck may be the user's choice of application, the application itself (such as older versions), or a configuration problem on the remote system. Creating unique combinations as flow identifiers can get tricky because often a remote host can have multiple IP addresses, or even have a pool of IP addresses that are assigned round-robin in DNS. In this table, the hostname is defined instead of each unique IP address associated with it. This minimizes the number of case records that have to be maintained and provides a balance that best fits network engineers' ability to manage cases.

Table 4. The flowMapping table setup.

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_increment
user	varchar(25)	YES	MUL	NULL	
remoteHost	varchar(50)	YES	MUL	NULL	
assigned_to	varchar(25)	YES	MUL	NULL	
app	varchar(10)	YES	MUL	NULL	
status	tinyint(3) unsigned	YES	MUL	NULL	

flowMapping field descriptions:

- *id* – unique ID for each entry
- *user* – username of the person transferring data
- *remoteHost* – hostname of the remote system
- *assigned_t*o – engineer working on the case
- *app* – application being used to transfer data
- *status* – current status of the case

The Status Table

The case status value from the flowMapping table maps directly to a separate status table, which converts an integer value to a message:

Table 5. The status table setup.

Field	Type	Null	Key	Default	Extra
id	tinyint(3) unsigned	NO	PRI	0	
message	varchar(50)	YES		NULL	

This table can be initially populated with information when the tables are setup since the case status ID values do not change. Below is an example of some types of status messages that can be used. The *id* field values are what appear in the flowMapping table described in the previous section.

Table 6. Sample case status messages.

id	message
1	new
2	assigned
3	user contacted
4	pending
5	closed - problem resolved
6	closed - user not interested
7	other - see note
8	inactive

The flowNotes Table

In order to have a ticketing-type system, notes need to be entered about each case. Here, the notes entered by engineers are stored in the flowNotes table and are mapped to the unique ID in the flowMapping table, which defines a case (based on the username:remotehost:application set).

Table 7. The flowNotes table setup.

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_increment
stamp	datetime	YES		NULL	
mapID	int(10) unsigned	YES	MUL	NULL	
staff	varchar(15)	YES		NULL	
message	mediumtext	YES		NULL	

flowNotes field descriptions:

- *id* – unique ID for each entry
- *stamp* – timestamp of the entry
- *mapID* – this field maps to the *id* field in the flowMapping table, which ties it into the case
- *staff* – the engineer that entered the note
- *message* – the note entered

The flowHistory Table

The flowHistory table is used to track the state change of flows, such as when the flow was added to the database as well as any change of status involving the cases, and also logs the engineer who worked on it and when the change was made. This information is necessary for calculating before-and-after metrics, or for seeing when the last update was made to a case. This table is almost identical to the flowNotes table but serves a different purpose.

Table 8. The flowHistory table setup.

```
+---------+------------------+------+-----+---------+----------------+
| Field   | Type             | Null | Key | Default | Extra          |
+---------+------------------+------+-----+---------+----------------+
| id      | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| stamp   | datetime         | YES  |     | NULL    |                |
| mapID   | int(10) unsigned | NO   | MUL | 0       |                |
| staff   | varchar(25)      | YES  |     | NULL    |                |
| message | varchar(255)     | YES  |     | NULL    |                |
+---------+------------------+------+-----+---------+----------------+
```

flowHistory field descriptions:

- *id* – unique ID for each entry
- *stamp* – timestamp of the entry
- *mapID* – this field maps to the *id* field in flowMapping, which ties it into the case
- *staff* – the engineer that made the status change or whether it was added by the automated backend scripts
- *message* – detail about the status change

III. Obtaining Data

The collection of the network flow data is done piecemeal and depends on the methods selected. As previously mentioned, the application and Argus options were chosen for this implementation due to the network environment. Perl is the language-of-choice because of its ability to easily parse log files with regular expressions and its decent MySQL support.

Application Data

The file transfer applications employed by users are modified to log specific information about the transfers in each system's syslog file. The actual code used to accomplish this is outside the scope of this whitepaper, but since most file transfer applications already calculate the throughput rate and know the username and file location, it can be as simple as executing a logger command.

Once data is logged in the syslog file, a Perl script can be run as often as necessary (daily is usually sufficient) to extract each line and parse through the fields to retrieve their values. In environments where there is a centralized syslog collector, syslog data can be processed in a single location and in real-time. In order to minimize flow records stored in the database, small-sized flows should be filtered out. Small files often transfer so fast that TCP doesn't have time to reach its maximum rate and calculation of the rate can be greatly skewed. In addition, you generally don't want local transfers to be included, so anything coming from a local IP address can also be discarded. Make sure that the correct data conversions are used since applications often record rates in bytes instead of bits.

A simple SQL insert statement can be used to enter all the data into the flowData table. The unique constraints on the tables should automatically avoid any duplicate entries if syslog files aren't rotated. An example of this statement is:

```
insert ignore into flowData values ("", "2010-01-28 07:15:00", "hostA",
"bbftpd", "GET", "userName", 1755463680, 79.95, 167.5,
inet_aton("192.168.1.1"), "remoteHost.foo.com", "/path/to/filename/foobar");
```

While the script is processing data and inserting it into the database, it's also useful to check each flow record for performance issues so they can be easily identified and alert notices can be sent to engineers. A general algorithm is applied to each new record and, if it meets the criteria, the record is added as a new flow case and a notification email is sent to the engineers. An example of a potential problem flow may be a file that is over 100M, takes more than 10 minutes to transfer, and gets less than 100 Mbps.

A quick check of the flowMapping table can show whether the same combination of user, remote system, and application already exists as a case and, if so, which engineer is assigned to it. If a flow is tagged as a new case, the flowMapping table should be updated with the information as follows:

```
insert into flowMapping values("", "userName", "remoteHost.foo.com", NULL,
"bbftpd", 1)
```

This statement sets the case to new, with no engineer assigned to it. The flowHistory table should also be updated to create a record of when the case was added and by which method:

```
insert into flowHistory values("", NOW(), $lastID, "automated", "userName on
remoteHost.foo.com using bbftpd was added");
```

In this example, the value for $lastID is the unique ID value returned from the flowMapping table insertion. This allows for a mapping between the flowHistory records and the flowMapping table.

Flow Detail Data

To extract the detailed information that is of interest for flow analysis, a script is needed to cluster the Argus flows and print out the values that will be inserted into the back-end database:

```
racluster -n -p 6 -z -Z b -s stime ltime state saddr sport dir daddr
dport proto dur flgs spkts dpkts sbytes dbytes load ploss swin dwin
sloss dloss tcpopt
```

An example of a flow record printed from the above command is shown below (printed as one line):

```
2011-10-13 23:50:11,2011-10-13 23:50:39,sSEfFR,192.168.1.1,60273, -
>,192.168.2.1,22,tcp,27.642387, e &
,7592,8678,23693035,558004,7018076.500000,0.000000,524280,10722304,0,0,16114,
1150,2052.836429,1943.671910,MwsS  T
```

As before, a lot of the resulting information should be filtered out. Unlike the application logging, however, there are records that are not specific to file transfers and all traffic will be returned. To minimize the size of the flowDetails database, only the remote hosts that currently have open cases are allowed to be entered. In addition, specific ports can be filtered out if necessary. Very small transfers are dropped again as well.

Basic automated troubleshooting can be performed on this data. Triggered alerts can be sent out for common identifiers of performance issues, such as:

- If packet loss (retransmits) exceeds some minimal percentage
- If the TCP send or receive window is a static size for all flows from the same system (often seen in older applications that don't support dynamic windowing)
- If the TCP send or receive window is less than a couple of megabytes in size (this varies depending on expected bandwidth and latency)

- If certain TCP options are not being set or negotiated for use, such as selective ACK, ECN, or timestamps
- If the size of the file transfer is greater than a gigabit and the application (as determined by the service port) being used is only single stream

If Web100 is used to collect additional information, then the optimal bandwidth delay product can easily be calculated and compared to what's being used. There are many options, depending on the various flow characteristics that are captured.

IV. Front-End Interface

With all the data collected, a front-end interface is needed to easily manage and present the information in a way that is useful to engineers for analysis. A web-based front end allows for centralized, cross-platform access to the data. For the front end presented here, the PHP programming language is used to query the database directly to retrieve flow records, and provides powerful functionality for processing web data. The front-end website includes the following components for viewing and updating various aspects of the flow cases: an "interesting flows" section, a work-flow section, a remote host search function, a site-matching function, and a section for reporting, tracking, and updating the status of cases.

Interesting Flows

The term "interesting flows" refers to flow records that were previously tagged as being potentially limited. The back-end script that processes the flow data generally tags the flows and inserts them into the flowMapping table. Alternatively, any existing record can be tagged by an engineer via the website, which then adds it to the interesting flow list and displays it in a separate section of the website for easy referral.

To obtain a list of interesting flow cases currently open, along with some basic information about who is working on the flow, its status, and the associated user, application, and remote host, a query joining several tables is needed:

```
select a.id, a.user, a.remoteHost, a.assigned_to, a.app, a.status, b.message,
max(c.stamp) as lastUpdate, max(d.stamp) as lastFlow,
UNIX_TIMESTAMP(max(d.stamp)) as lastStamp, UNIX_TIMESTAMP(NOW()) as
currentStamp, inet_ntoa(d.remoteIP) as remoteip from (((flowMapping a join
status b on a.status=b.id) left join flowNotes c on c.mapID=a.id) left join
flowData d on a.remoteHost=d.remoteHost and a.user=d.user and a.app=d.app)
where a.status < 5 group by a.id order by $sort,a.remoteHost
```

The data can be sorted in different ways, as specified by the $sort variable. For a large number of flows, it may be most useful to sort by engineer, to make it easier to keep track of all the open cases under their name. It's not unusual for there to be long periods of inactivity between data transfers, so engineers can easily focus on active flows. Each field should be linked to more detailed information. For example, clicking on the user's name will bring up data flows from that user from any location. If implementing a ticket-like system, clicking on the status of a case should link to a detailed section about the specific activity involving that case.

Work Flow

The work-flow area of the front end displays all activity associated with a case and is where engineers make updates. The web page should have an area where engineers can assign the case to themselves or others, select case status, and add any notes. Any changes made will log a message to the flowHistory table along with a timestamp to automatically capture the history of work on each case. All the notes associated with a case are stored in a separate flowNotes table. The web page should display both the notes and the history so that engineers can easily see the current status of the work done. The *mapID* field in both the flowHistory and flowNotes tables maps directly to the unique *id* field in the flowMapping table, making it easy to extract the respective records by using commands such as:

```
select stamp, staff, message from flowNotes where mapID=$id order by stamp
desc

select stamp, staff, message from flowHistory where mapID=$id group by stamp
order by stamp desc limit 20
```

In the above commands, $id is the unique *id* field of the case from flowMapping. It's easiest to just pass this variable along as a POST value in the PHP code.

In environments that have a production ticketing system, the user's contact information, such as email or phone number, can often be pulled directly from the systems so that engineers don't need to track down how to contact the user.

It may also be useful to list all the slow flow records associated with a case to show how consistent the problem is over time. The algorithm shown below is the same as the one the back-end data collector uses to tag potential problem flows.

```
select a.user, a.remoteHost, a.app, b.stamp, b.host, b.direction,
round((b.size/1024/1024),2) as megabytes, round((b.length/60),2) as time,
b.file, b.mbps from flowMapping a, flowData b where a.remoteHost=b.remoteHost
and a.user=b.user and b.mbps < $mbps and b.length > $time and b.size > $size
and a.app=b.app and a.id=$id order by stamp desc limit 50
```

The variables for mbps, duration, and size can be pre-defined based on what application is being used. Multi-stream applications will generally have faster rates than single-stream applications and should have separate parameters defined. As before, the flowMapping *id* field value is passed along via a POST- or GET-type method, or is included in the URL for retrieval via $HTTP_GET_VARS[id].

Remote Host Search

In order to quickly find information about a specific remote system, a search or drop-down selection list should be available. Under this section, all flow information for a particular host is listed along with the users who have transferred to or from the system, the host system's points of contact, and what current or past cases have been associated with it.

To get a quick drop-down menu for selecting a remote host, one can make a select query to the flowData as follows:

```
select remoteHost,inet_ntoa(remoteIP) from flowData group by
remoteHost,remoteIP order by remoteHost
```

From the user selection, contact information about the system administrator or other site contact can be pulled and ordered by remote host name. The user list is also important so engineers know whom to work with directly. Again, this information can be fetched by the remote host using the table that is populated with the application log data, which can include the username:

```
select distinct user from flowData where remoteHost="$remoteHost" or
remoteIP=inet_aton("$remoteIP") order by user
```

The variables for this query are retrieved from the drop-down menu that was created above or from inputs entered into a search text box on the website.

Another quick query will also gather information on whether or not a particular remote system has had any cases open in the past, or has any cases currently being worked on by an engineer:

```
select a.id, a.user, a.assigned_to, a.app, b.message from flowMapping a,
status b where b.id=a.status and remoteHost="$remotehost" order by user
```

Now the actual flow records come into play. The database has two tables for flow records: the flowData table from application logging, and the flowDetails table containing more comprehensive information from network taps. The actual application logs are easier to work with since multi-stream flows are aggregated and the user is shown. To retrieve selective flow information about a specific remote host, as defined in the drop-down menu or search box, the following query can be made:

```
select inet_ntoa(remoteIP) as remoteip, stamp, host, app, direction, user,
round((size/1024/1024),2) as megabytes, round((length/60),2) as minutes,
mbps, remoteHost, file from flowData where remoteIP=inet_aton("$remoteIP")
order by stamp desc limit $limit,101
```

Since there are probably a large number of records in the flow database, it is best to search by the IP address as represented in integer form. The $limit variable is used to offer paging so that not all the records are returned at once. The *select* query returns the remote system, user, the local system involved in the transfer, the application being used, the direction of the transfer, file size, duration, transfer rate, and if available, the path and location of the file.

To fetch the detailed flow records, a bit more logic has to be applied. Both source and destination should be queried since network collectors don't always correctly record the initiating host.

```
select distinct b.id, b.stime, b.etime, inet_ntoa(b.destIP) as remoteIP,
inet_ntoa(b.srcIP) as localIP, b.srcport, b.dstport,
round((b.dbytes/1024/1024),2) as dbytes, round((b.sbytes/1024/1024),2) as
sbytes, b.mbps, b.loss, b.swin, b.dwin, b.srtx, b.drtx, b.timestamps, b.sack,
b.sackOK from flowData a, flowDetails b where a.remoteIP=b.destIP and
a.remoteIP=inet_aton("$remoteIP")  UNION select distinct b.id, b.stime,
b.etime, inet_ntoa(b.destIP) as remoteIP, inet_ntoa(b.srcIP) as localIP,
b.srcport, b.dstport,  round((b.dbytes/1024/1024),2) as dbytes,
round((b.sbytes/1024/1024),2) as sbytes, b.mbps, b.loss, b.swin, b.dwin,
b.srtx, b.drtx, b.timestamps, b.sack, b.sackOK from flowData a, flowDetails b
where a.remoteIP=b.srcIP and a.remoteIP=inet_aton("$remoteIP") order by stime
desc limit $limit,101
```

With the output from this query, it should be easy to identify the most common causes of slow performance: packet loss and small TCP window sizes. If the window sizes are static, then there could either be a problem with the application using static window sizes instead of using getsockopt() calls,

or a system tuning issue where dynamic windowing is disabled. System and application tuning problems can easily have an automated alert sent either to the user or to one of the network engineers. The same is true if retransmitted packets are over a certain percentage, relative to the amount of data being transferred.

One can also search by local system, user, or even application. This drill-down method facilitates the engineer's ability to find exactly the information needed based on various criteria.

Site Matching

Another useful feature is the ability to break down flows by origin and sub-domain. For example, if all flows from a particular university or organization are achieving the same transfer rate, then there's probably a limitation that is specific to the entire site, such as a firewall or over-subscribed network link. If only a couple of systems for a site are experiencing slowness, then the issue may be due to application or system tuning. From this information, it's easy to see the number of flows per site, the systems involved in data transfers, and all the users associated with the site. It helps to focus troubleshooting efforts on locations that would receive the most benefit.

A single SQL query can retrieve this information in a compartmentalized manner for display:

```
select substring_index(remoteHost, '.', -2) as domain, count(id) as total,
group_concat(distinct user order by user separator '<br>') as users,
group_concat(distinct remoteHost order by remoteHost separator '<br>') as
hosts from flowData where remoteHost not rlike '[0-9]+.[0-9]+.[0-9]+.[0-9]'
group by domain
```

This will break down all the existing flow records and group them by origin or site. The webpage will show who the users are at each site, the total number of flows to or from the site, and which systems are being used to transfer data. This meta representation of data allows engineers to quickly see who and what may be impacted by a slowness problem or network change. Additionally, each value displayed on the website is linked via a URL to pull up additional records as described previously (under Remote Host Search).

Reporting, Tracking, and Status Updates

It may be useful maintain a web page that lists TopN-type reports for things such as the top remote hosts based on cumulative data transferred or total number of flows. The MySQL sum() function makes this simple:

```
select id,remoteHost, round(sum(length/60/60),2)*1 as hours,
round(sum(size/1024/1024/1024),2) as totalbytes from flowData group by
remoteHost,remoteIP order by totalbytes desc limit 10
```

To keep up-to-date on the work done by engineers and any status changes to the flow cases, a separate page can be made to show all the notes added within the last week or any other user-selectable time period. The flowHistory table keeps a record of any state changes to a case, whether it is newly added, changed to a new status, or assigned to an engineer. For organizations with a large number of flows, this feature helps to efficiently track flow analysis efforts and work loads.

One other useful feature that can be implemented is monitoring for when a user becomes active again, or when a case that has been resolved starts having problems. Often this happens due to a system being rebuilt and not re-tuned as before, or sometimes a network upgrade has a negative impact on performance. Similarly to the "Interesting Flows" page, this area captures flows that match certain performance criteria, but specifically lists cases that have either already been resolved or set to inactive by an engineer. If the last flow that meets the parameters defined for poor performance is more recent than when the case was closed out or last updated, it can be visually highlighted on the page. This enables engineers to easily see when a problem might have re-occurred and requires additional troubleshooting.

V. Additional Options

A variety of additional options can also be incorporated into the flow database and analysis tool if appropriate for a given network environment. Additional options discussed below include incorporating NDT server data, NPAD reports, system administrator mappings, and data visualizations for easier analysis.

Tying Into an NDT Server

If you have an NDT server running on your network, you may want to include performance test results in your database. When a remote system runs a performance test, the web100srv process creates a nice key:value pair for each parameter monitored and adds a line to the system's /var/log/messages log file that looks like this:

```
Aug 17 15:06:05 localhost web100srv:
client_IP=10.1.1.1,c2s_spd=36757,s2c_spd=47258,Timeouts=0,SumRTT=296629,Count
RTT=3214,PktsRetrans=0,FastRetran=1,DataPktsOut=46253,AckPktsOut=0,CurrentMSS
=1460,DupAcksIn=33,AckPktsIn=3247,MaxRwinRcvd=23576576,Sndbuf=33554432,MaxCwn
d=3651460,SndLimTimeRwin=0,SndLimTimeCwnd=2909807,SndLimTimeSender=8162559,Da
taBytesOut=67565480,SndLimTransRwin=0,SndLimTransCwnd=64,SndLimTransSender=64
,MaxSsthresh=95681100,CurrentRTO=292,CurrentRwinRcvd=23576576,link=0,mismatch
=0,bad_cable=0,half_duplex=0,congestion=1,c2sdata=9,c2sack=9,s2cdata=8,s2cack
=9,CongestionSignals=1,PktsOut=46253,MinRTT=91,RcvWinScale=13
```

A separate table is created in the flows database that stores the relevant information from the log file entries as described above. See the Appendix for details on creating this table. The NDT reports are straightforward and easy to incorporate into the flow tools. A cron job is used to parse the log file daily and insert any new records. It can also email out alerts with the information. When using the front-end interface to pull up flow data for a particular remote host, the NDT records can be searched by IP address to see if that system has any previous performance tests.

Tying NPAD Reports Into the Database

For sites running their own NPAD server, performance test analyses can also be included with the other flow data used for troubleshooting. NPAD is similar to the NDT tests, but its testing is much more exhaustive and doesn't have a standardized report mechanism in key:value pairs.

When a remote system initiates an NPAD test, the results are saved to an HTML file under ServerData/Reports-YYYY-DD/<ip_addr>:timestamp.html. The contents of the files vary greatly from test to test and there is no easy way to parse through them. A customized script needs to be written to extract key information such as the maximum throughput, loss, window scale, and calculated BDP (bandwidth delay product, discussed earlier). It may be easier just to retrieve the IP address of the remote host and be able to pull up the entire HTML file for review by an engineer.

An example table setup is included in the appendix. Since each report varies, not all fields will have results. Aside from key values, the HTML report is saved in the database for review when working with a particular remote system.

System Administrator Mapping

It is important to keep a record of all the points of contact for each system or site for easy reference. Another table in the database can easily store a mapping of an administrative contact for each system. When a search is performed via the front-end interface, results can show who the administrator is for that particular system, saving the engineers time.

Visualization

With all the flow records stored in a database, creating a graphical representation of the data can greatly aid in troubleshooting and allow engineers to easily see performance trends over time. Visualization enables engineers to select specific characteristics of a flow that can help identify issues that may easily be missed by looking at raw data. Figure 3 shows an example of a flow data visualization.

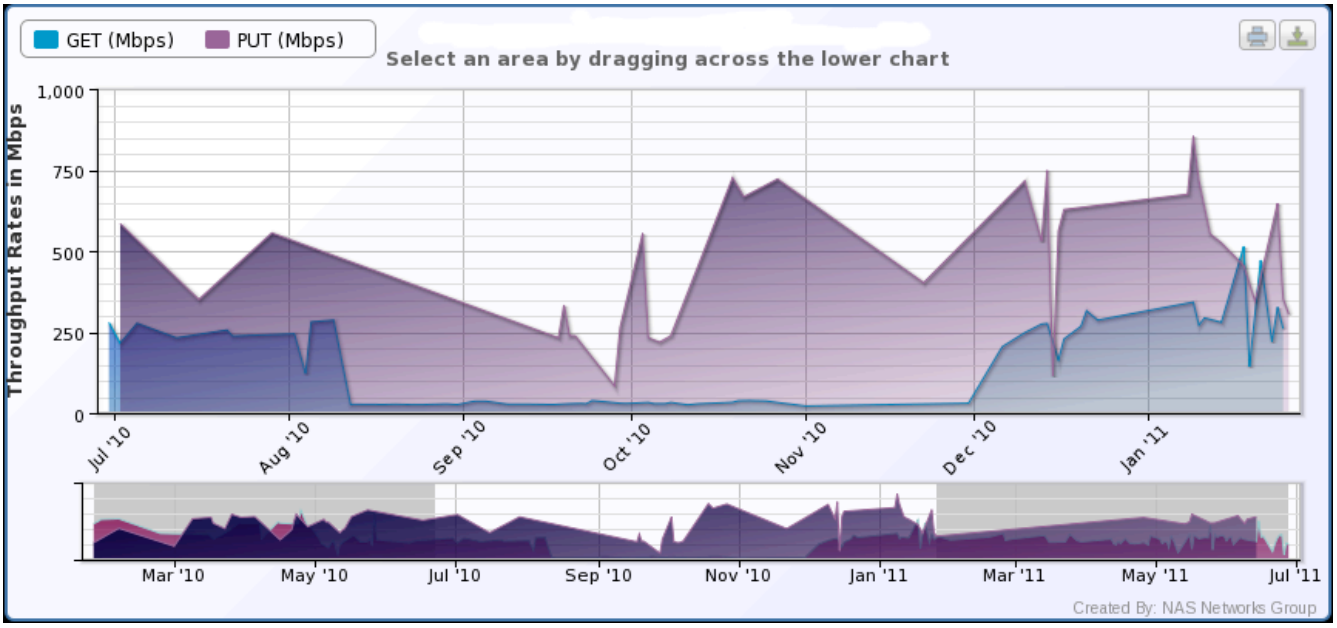


Figure 3. Visualization of throughput over time for a host, showing the unexpected directional impact of a network change.

There are many programming tools available for generating graphics, and it is left to the reader to determine which software toolkits to use. However, it is important to be able to identify what data is most useful to graph. Some key characteristics to visualize are:

- Remote host or IP address
- Source host or IP address (is the problem specific to a local system?)
- Site (is it a site-wide slowness affecting all systems within a domain?)
- Direction (is the slowness only in one direction?)
- Application (is it an application-specific issue?)

In addition, the data can be represented in different ways, such as:

- Average rate for each day
- Maximum rate for each day
- Minimum rate for each day
- Standard deviation of rates for each day
- Total data transferred per day

For systems that often have high deviations, there may be a resource contention problem, either with the network or the system itself. It may be useful to also graph details such as number of retransmits to see if it the problem is specific to an entire site. As more data is collected over time, better historical analyses can be made and engineers can more easily determine what is typically expected from a remote system.

VI. Conclusion

Using tools to proactively and automatically identify transfers with below-average performance rates is a simple technique to aid engineers in their analysis work. Removing the effort of collecting tcpdumps or other such manual methods gives engineers more time to focus on analyzing the data, rather than coordinating with users or obtaining accounts on remote systems to run tests for themselves. Because many of the most common performance problems are easy to identify given the correct set of data, a lot of the analysis work can be automated through the use of these tools, further saving engineers time spent in diagnosing problems.

Since most sites already deploy at least one method of data collection, a little extra time invested up-front to prepare an integrated collection and management tool for flow analysis can benefit both network engineers and users almost immediately. Many users will achieve faster transfer rates with their data and engineers will see improvements to their overall network utilization. With further development, the data collection and notification can provide real-time flow analysis to give users and engineers immediate feedback.

VII. Appendix

Appendix A – MySQL Table Creation

```
CREATE TABLE `flowData` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `stamp` datetime default NULL,
  `sourceHost` varchar(25) default NULL,
  `app` varchar(25) default NULL,
  `direction` char(3) default NULL,
  `user` varchar(25) default NULL,
  `fileSize` bigint(20) unsigned default NULL,
  `duration` float unsigned default NULL,
  `mbps` float unsigned default NULL,
  `remoteIP` int(10) unsigned default NULL,
  `remoteHost` varchar(50) default NULL,
  `file` varchar(255) default NULL,
  PRIMARY KEY  (`id`),
  UNIQUE KEY `stamp` (`stamp`,`file`),
  KEY `remoteHost` (`remoteHost`),
  KEY `user` (`user`),
  KEY `app` (`app`),
  KEY `sourceHost` (`sourceHost`),
  KEY `remoteIP` (`remoteIP`)
) ENGINE=MyISAM AUTO_INCREMENT=2390799 DEFAULT CHARSET=latin1 |


CREATE TABLE `flowDetails` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `stime` datetime default NULL,
  `etime` datetime default NULL,
  `destIP` int(10) unsigned default NULL,
  `srcIP` int(10) unsigned default NULL,
  `srcport` int(10) unsigned default NULL,
  `dstport` int(10) unsigned default NULL,
  `dbytes` int(10) unsigned default NULL,
  `sbytes` int(10) unsigned default NULL,
  `mbps` float unsigned default NULL,
  `loss` float unsigned default NULL,
  `swin` int(10) unsigned default NULL,
  `dwin` int(10) unsigned default NULL,
  `srtx` int(10) unsigned default NULL,
  `drtx` int(10) unsigned default NULL,
  `timestamps` enum('N','Y') default NULL,
  `sack` enum('N','Y') default NULL,
  `sackOK` enum('N','Y') default NULL,
  PRIMARY KEY  (`id`),
  UNIQUE KEY `stime` (`stime`,`destIP`,`srcport`),
  KEY `remoteIP` (`remoteIP`)
) ENGINE=MyISAM AUTO_INCREMENT=116785 DEFAULT CHARSET=latin1 |
```

```
CREATE TABLE `flowMapping` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `user` varchar(25) default NULL,
  `remoteHost` varchar(50) default NULL,
  `assigned_to` varchar(25) default NULL,
  `app` varchar(10) default NULL,
  `status` tinyint(3) unsigned default NULL,
  PRIMARY KEY  (`id`),
  UNIQUE KEY `user` (`user`,`remoteHost`,`app`),
  KEY `status` (`status`),
  KEY `remoteHost` (`remoteHost`),
  KEY `app` (`app`),
  KEY `assigned_to` (`assigned_to`)
) ENGINE=MyISAM AUTO_INCREMENT=434 DEFAULT CHARSET=latin1 |


CREATE TABLE `status` (
  `id` tinyint(3) unsigned NOT NULL default '0',
  `message` varchar(50) default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 |


CREATE TABLE `flowNotes` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `stamp` datetime default NULL,
  `mapID` int(10) unsigned default NULL,
  `staff` varchar(15) default NULL,
  `message` mediumtext,
  PRIMARY KEY  (`id`),
  KEY `mapID` (`mapID`)
) ENGINE=MyISAM AUTO_INCREMENT=705 DEFAULT CHARSET=latin1 |


CREATE TABLE `flowHistory` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `stamp` datetime default NULL,
  `mapID` int(10) unsigned NOT NULL default '0',
  `staff` varchar(25) default NULL,
  `message` varchar(255) default NULL,
  PRIMARY KEY  (`id`),
  KEY `mapID` (`mapID`)
) ENGINE=MyISAM AUTO_INCREMENT=17383 DEFAULT CHARSET=latin1


CREATE TABLE `sysadmin` (
  `id` smallint(5) unsigned NOT NULL auto_increment,
  `remoteHost` varchar(50) default NULL,
  `adminName` varchar(50) default NULL,
  `adminPhone` varchar(25) default NULL,
  `adminEmail` varchar(50) default NULL,
  `adminNote` varchar(200) default NULL,
  PRIMARY KEY  (`id`),
  KEY `remoteHost` (`remoteHost`)
) ENGINE=MyISAM AUTO_INCREMENT=25 DEFAULT CHARSET=latin1
```

```
 CREATE TABLE `flowNDT` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `stime` datetime default NULL,
  `remoteIP` int(10) unsigned default NULL,
  `c2sSpeed` int(10) unsigned NOT NULL,
  `s2cSpeed` int(10) unsigned NOT NULL,
  `timeouts` int(10) unsigned NOT NULL,
  `sumRTT` int(10) unsigned NOT NULL,
  `countRTT` int(10) unsigned NOT NULL,
  `pktsRetrans` int(10) unsigned NOT NULL,
  `fastRetrans` int(10) unsigned NOT NULL,
  `dataPktsOut` int(10) unsigned NOT NULL,
  `ackPktsOut` int(10) unsigned NOT NULL,
  `mss` smallint(5) unsigned NOT NULL,
  `dupAcksIn` int(10) unsigned NOT NULL,
  `ackPktsIn` int(10) unsigned NOT NULL,
  `maxRwinRcvd` int(10) unsigned NOT NULL,
  `sndbuf` int(10) unsigned NOT NULL,
  `maxCwnd` int(10) unsigned NOT NULL,
  `sndLimTimeRwin` int(10) unsigned NOT NULL,
  `sndLimTimeCwnd` int(10) unsigned NOT NULL,
  `sndLimTimeSend` int(10) unsigned NOT NULL,
  `dataBytesOut` bigint(20) unsigned NOT NULL,
  `sndLimTransRwin` int(10) unsigned NOT NULL,
  `sndLimTransCwnd` int(10) unsigned NOT NULL,
  `sndLimTransSender` int(10) unsigned NOT NULL,
  `maxSsthresh` int(10) unsigned NOT NULL,
  `currentRTO` int(10) unsigned NOT NULL,
  `rwinRcvd` int(10) unsigned NOT NULL,
  `link` smallint(5) unsigned NOT NULL,
  `mismatch` tinyint(3) unsigned NOT NULL,
  `badCable` tinyint(3) unsigned NOT NULL,
  `halfDuplex` tinyint(3) unsigned NOT NULL,
  `congestion` int(10) unsigned NOT NULL,
  `c2sdata` tinyint(3) unsigned NOT NULL,
  `c2sack` tinyint(3) unsigned NOT NULL,
  `s2cdata` tinyint(3) unsigned NOT NULL,
  `s2cack` tinyint(3) unsigned NOT NULL,
  `congestionSignals` tinyint(3) unsigned NOT NULL,
  `pktsOut` int(10) unsigned NOT NULL,
  `minRTT` int(10) unsigned NOT NULL,
  `rcvWinScale` tinyint(3) unsigned NOT NULL,
  PRIMARY KEY  (`id`),
  UNIQUE KEY `stime_2` (`stime`,`remoteIP`),
  KEY `remoteIP` (`remoteIP`)
) ENGINE=MyISAM AUTO_INCREMENT=38 DEFAULT CHARSET=latin1
```

```
CREATE TABLE `flowNPAD` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `stime` datetime default NULL,
  `remoteIP` int(10) unsigned NOT NULL,
  `remoteHost` varchar(40) default NULL,
  `url` varchar(50) NOT NULL,
  `targetRate` varchar(10) default NULL,
  `targetRTT` smallint(5) unsigned NOT NULL,
  `actualRTT` float unsigned NOT NULL,
  `mss` smallint(5) unsigned NOT NULL,
  `wscale` tinyint(3) unsigned NOT NULL,
  `sack` tinyint(3) unsigned NOT NULL,
  `timestamps` tinyint(3) unsigned NOT NULL,
  `maxMbps` float unsigned NOT NULL,
  `loss` float unsigned NOT NULL,
  `bdp` int(10) unsigned NOT NULL,
  `tcpTest` enum('Pass','Fail','Warning') default NULL,
  `dataTest` enum('Pass','Fail','Warning') default NULL,
  `lossTest` enum('Pass','Fail','Warning') default NULL,
  `bufferTest` enum('Pass','Fail','Warning') default NULL,
  `pathTest` enum('Pass','Fail','Warning') default NULL,
  `report` text,
  PRIMARY KEY  (`id`),
  UNIQUE KEY `stime` (`stime`,`remoteIP`),
  KEY `remoteIP` (`remoteIP`),
  KEY `remoteHost` (`remoteHost`)
) ENGINE=MyISAM AUTO_INCREMENT=50 DEFAULT CHARSET=latin1
```

Appendix B – Additional Information

To find additional information about some of the software mentioned in this paper, please visit the following sites:

Argus – http://www.qosient.com
MySQL – http://mysql.com
Perl – http://www.perl.org
Apache – http://www.apache.org
PHP – http://php.net
Web100 – http://web100.org
Web10G – http://web10g.org