The Traffic Matrix

Martin van den Nieuwelaar

A Traffic Measurement tool for the AUCS network

I. Introduction

AUCS[1] Communications Services is a company that provides Voice, Data, and Internet services to a largely European customer base. The Internet division supplies European and Transatlantic Internet transit for large businesses and Internet providers. The Internet backbone network spans 14 countries in Europe alone.

The complete AUCS network has until now been monitored solely by SNMP gathered statistics, with subsequent visualisation using MRTG. This system, while useful, is limited to a fairly coarse level of granularity. To gain a better understanding of traffic flow across the AUCS Internet backbone, The Traffic Matrix has been developed. It gives a significant advantage to the capacity planners and network designers. With it, we can view AS (Autonomous System) to AS traffic flows, with a view on building a network that gives customers better performance, while minimising wasted bandwidth.

The Traffic Matrix is still in development and is an evolving beast. What we have achieved thus far is a system that provides useful real-world traffic flow information. This is based on a combination of both passive and active network measurements. A secondary goal of the project is to have the ability to simulate flow and network changes. These are still under development.

II. DESIGN & IMPLEMENTATION

AUCS runs a Cisco Powered Network, ie. one constructed exclusively using Cisco equipment. Routers are typically 7000 series at the access layer, with 12000 series used in a number of core locations. Inter-POP circuit speeds vary between 45 and 155 Mbps.

Preliminary investigation of statistics-gathering tools revealed few suitable candidates. Generally tools fell into two broad categories.

1. Based on SNMP stats. These would be able to provide us with interface-level information, but with no finer level granularity.

2. Based on PC hardware where the PC directly gathers statistics off of a directly connected interface – ala NeTraMet. In such situations the PC is connected to a medium such at Ethernet where it can eavesdrop on passing information. This would not be suitable in our case as our network connections are point to point. The possibility of splicing such statistics gathering devices into our network was briefly investigated but quickly dismissed.

One other category presented itself as being a more interesting candidate. Netflow Exports are flow statistics produced by certain models of Cisco equipment. They have the advantage of providing a wide range of measurements, including the AS to AS traffic flows we are interested in. The disadvantage is that they are a proprietary Cisco design and are therefore only available on Cisco equipment, and in fact only on certain models. Fortunately the AUCS network contains only Cisco routers, so we decided to pursue this avenue further.

Diagram showing contents and sizes of Netflow Export packets as produced by a router

Investigation of tools that work with Netflow Exports revealed two candidates. The first being the Cisco proprietary collector, and second a public domain tool written by Daniel McRobb called Cflowd. Comparing the two, I found Cflowd to provide a simple elegant solution, with the benefit of the code being easily extensible.

Cflowd is a package that runs under UNIX. It works as a daemon, continually gathering Netflow Exports that have

Martin is employed as a systems architect in the planning and design department of AUCS Communications Services B.V. He can be reached via E-Mail at martin.nieuwelaar@aucs-europe.com. See also the company website http://www.aucs-europe.com.

[1] Pronounced letter by letter “A”, “U”, “C”, “S”, and not “arcs” or something rhyming with “hawks”.

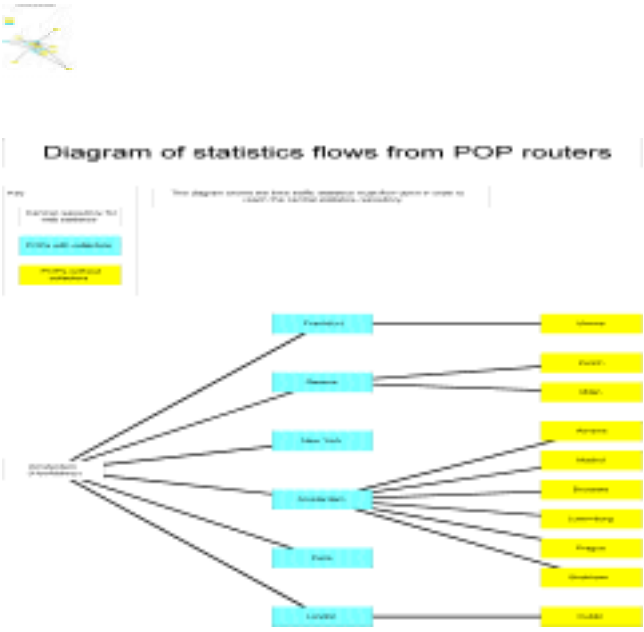
been exported by an associated Cisco router. Cflowd uses aggregation of statistics to reduce the huge amount of statistical information produced by a heavily utilised router. It also stores its data in a spatially compact format.

The plan was to use Cflowd running on machines to collect statistics at various points in our network, and to relay the stats back to a central location for post-processing and report generation.

A. *Where to place the collectors*

At the start of the Traffic Matrix project the AUCS network approximated that of a star network topology with the redundant core in Amsterdam.

Where to collect the statistics in the network was of course an important question. Router statistics could be sent from each POP back to a central collector in Amsterdam. This would of course minimise the number of collectors, however there would have to be a large amounts of long-haul bandwidth being used for the transportation of the statistics. The other extreme would be to place collectors in every POP. This would require an excessively large number of collectors. We decided to reach a compromise by placing collectors in our larger POPs. Statistics collected from smaller POPs would be picked up at the nearest POP with a collector.



B. *Choosing the collector hardware*

After deciding on the locations for collectors, it was necessary to work out the hardware specification for the units. One of the main problems was that we had no real idea how much CPU power would be required in order to collect statistics from possibly multiple backbone routers. There were only approximate guidelines available from the author of Cflowd. These were qualified with doses of "your mileage may vary" qualifications.

It would have been possible to try a number of machines out in the field just to see how they performed, but the deployment of such machines would take quite some time. Instead, I decided to simulate backbone routers in the laboratory by using a “Netflow Export simulation tool”. With this approach we would be able to compare different hardware performance directly. With such a tool, Netflow Export generation would be precisely controlled, and more importantly, repeatable. This is something we would not be able to achieve in the field.

Writing the Netflow Export simulator turned out to be quite straightforward. It involved routines to both generate the export packets, and to export them at a controlled rate. The Netflow Export packet generation was based on pseudo-random numbers, thereby ensuring the repeatability of experimental runs. The second part to the simulator involved exporting the generated packets at a controlled rate. I wrote the simulator in C and ran it on a GNU/Linux system. A 233 MHz Pentium proved to be amply fast enough for the simulator. Ethernet would be the preferred method of connecting the collectors in the field, with 10 MHz minimum, but also 100 MHz available. In order to avoid problems with network congestion in the testing phase I decided to use 100 MHz Ethernet cards. The tests would also be run "back to back" with the simulator directly connected with a crossed Ethernet cable to the collector under study. Ethernet works well under such conditions as there are no Ethernet packet collisions to degrade performance. I did not expect any problems with the network being the bottleneck as Netflow Export flow rates were expected to be of the order of only 0.5-2 Mbps. This value was roughly estimated from other people in the Cflowd mailing list[2] who were gathering statistics on their own networks.

Deciding between an Intel-based or SPARC-based system was the main choice to be made. With this in mind I ran performance tests comparing an ULTRA 10 clone with UW-SCSI drive running Solaris against a Pentium 233 MHz PC with IDE drive running GNU/Linux.

Cflowd works as a number of processes. Two of these (cflowdmux and cflowd) are responsible for actual gathing of the statistics. A third (called cfdcollect) picks up the stats from cflowd and writes them to disk. It is possible to run cfdcollect on a separate machine from cflowd, however I run them on the same machine in order to improve the level of aggregation possible.

[2] Send E-Mail to cflowd-request@caida.org for joining instructions. The CAIDA site itself contains much useful information and can be reached at www.caida.org.

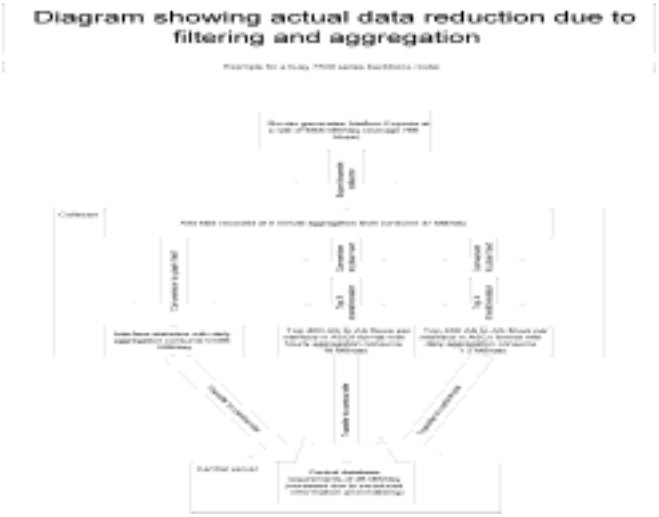
Judging by the hardware under comparison I expected the SPARC to outperform the PC by a huge margin. In fact this was not the case. I found the SPARC would run with virtually no load during the collection of the statistics, however as soon as cfdcollect started a write to disk, the CPU load would go to 100% and remain there for quite some time. In comparison the GNU/Linux box would maintain an overall lower and more consistent load.

Out of band management was also taken into consideration. Every box in the AUCS network has out of band management by way of a serial port manager. In case there is a serious problem, we can always connect to the unit in question via a secure OOB connection. A SPARC system can be administered entirely via this connection. At present a PC system cannot. GNU/Linux supports some level of OOB management via a serial port. In fact everything in the boot process from the "lilo" prompt onwards. I decided that this level of management was adequate. Based on price, performance and maintainability I made the decision to go with PCs running GNU/Linux was made. In actual fact, in the six months the collectors have been out in the field, it has not been necessary to use the OOB at all.

C. Aggregation and filtering - ways to reduce resource requirements

A very useful feature of the Cflowd software is the ability to perform aggregation. A typical router will produce Netflow Exports continuously. This stream may have bandwidth requirements of anywhere from a few kilobits per second up to several megabits per second. With a large number of routers in the network it was obvious that collecting every piece of statistical information was out of the question. The AUCS network contains over 100 routers, though it would be necessary to collect statistics from only a moderate portion of these.

Data-rate reduction is performed on each collector in a number of steps. Firstly, we ignore any Netflow Export packets that are uninteresting. We collect AS to AS information and interface statistics only. Secondly, flow information is aggregated over time. Cflowd is configured to write all data flows to disk every five minutes. At the end of each day, we additionally aggregate to the level of one hour. Hourly aggregation is coarse enough to significantly reduce the amount of data being recorded while still being fine enough to show trends in traffic usage throughout a particular day. Following this aggregation, we keep only statistics on the largest flows. There are a large number of extremely small flows that contribute very little useful information. This step further reduces data requirements.



The AS to AS statistics we gather are available as the output from Cflowd in the following form. For each router interface with statistics exporting enabled, we receive statistics regarding inward data transmission. That is to say the flow is unidirectional and only the inward flow is measured. For this flow, we receive source and destination AS information, as well as packets and bytes transferred. A *bytes per second* field is generated based on the lifetime of the flow.

The interface statistics on the other hand while unidirectional, contain ingress and egress interface information. This allows us to see the path traffic takes on our network. Information such as *bits per second* is also recorded for each interface flow statistic.

Of particular interest is knowing where a traffic flow (say a particular AS to AS flow) enters our network, and where it leaves. This will help us to get an idea of the requirements of our traffic and hopefully enable us to improve our network.

In order to be able to associate a flow with a source and destination city (or POP as that is also interesting), we need some sort of geographic association between our statistics and city (or POP). Each flow statistic contains the IP address of the router it was generated on. I have a hand-maintained mapping between router ip addresses and the city and POP where they are located. This is the only hand-maintained piece of information in the Traffic Matrix. Unfortunately I could think of no way to generate it automatically.

A second point of interest is on what router interfaces to gather statistics from. If we look at all router interfaces we will gather statistics for a particular data flow more than once (double-counting). This problem is solved by measuring on all (but only on) outward-pointing interfaces. That is to say interfaces pointing away from the AUCS network. This way we measure once and only once.

One interesting challenge in making sense of the statistics involves the interpretation of the interface information. For a particular router, the physical interface is represented by the *ifIndex* number. This number uniquely defines the physical interface on a router *at one point in time*. For example, Serial0/0/0 may have an *ifIndex* number of 3. The relationship between *ifIndex* number and the actual interface is an interesting one and worthy of discussion.

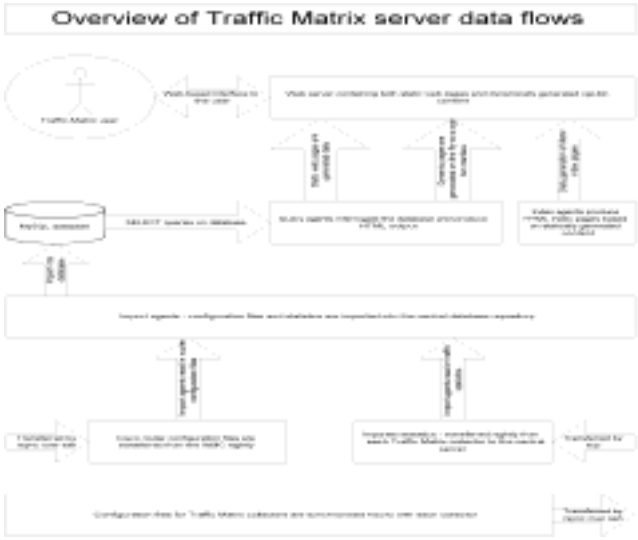
When the router is first powered on, *ifIndex* numbers are allocated to interfaces in an incremental fashion starting with slot 0 and ending at the highest populated slot. An interesting phenomena occurs when an interface card is added or removed from the system in a *hot-swap* fashion. Adding a new interface card in any slot of the router (even slot 0 for example) will result in the interfaces on that card being allocated the next highest available *ifIndex* numbers. Removing a card will result in that *ifIndex* number no longer being available. After a few cards have been added and removed the association between ifIndex number and the card position is not at all obvious. Even if one managed to keep track of the association, the next time the router is rebooted, all *ifIndex* numbers will be allocated in sequence again.

The varying *ifIndex* number problem was eventually solved by retrieving SNMP information from each router. This data contains the present association between *ifIndex* number and physical interface. Our INMC already retrieve the SNMP stats from each router on a daily basis, and so it was a straightforward exercise to import this data into a database table for the Traffic Matrix.

D. Central server

In order to produce network-wide reports, it is necessary to bring all collected statistics back to a central point. We have set up a SPARC ULTRA 1/200 MHz with 256 MB RAM and 30 GB of disk spread across multiple spindles. The machine runs Solaris 7. The software running on the server can be broken down into the following categories.

- Import agents
- MySQL database
- Query agents
- Web server + static content + cgi-bin scripts



Traffic statistics gathered on the collectors are transferred back to the central server on a daily basis. Import agents on the server process the statistics into a more suitable format. This may involve parsing router configuration files in order to produce the connection table for example where all inter-router connections are stored. It may also involve pre-calculation of fields in order to speed the response of cgi-bin scripts. After processing the statistics are imported into a MySQL database. Daily queries are run on the database in order to produce static reports. These are typically data-intensive reports that are generated early in the morning. *Light-weight* reports are generated on the fly by cgi-bin scripts. Such scripts are also used where the large number of potential reports would make it impossible to generate static reports for every combination. For example it would not be sensible to generate static reports for every traffic trend graph for each AS to every other AS each day.

E. Network Visualisation

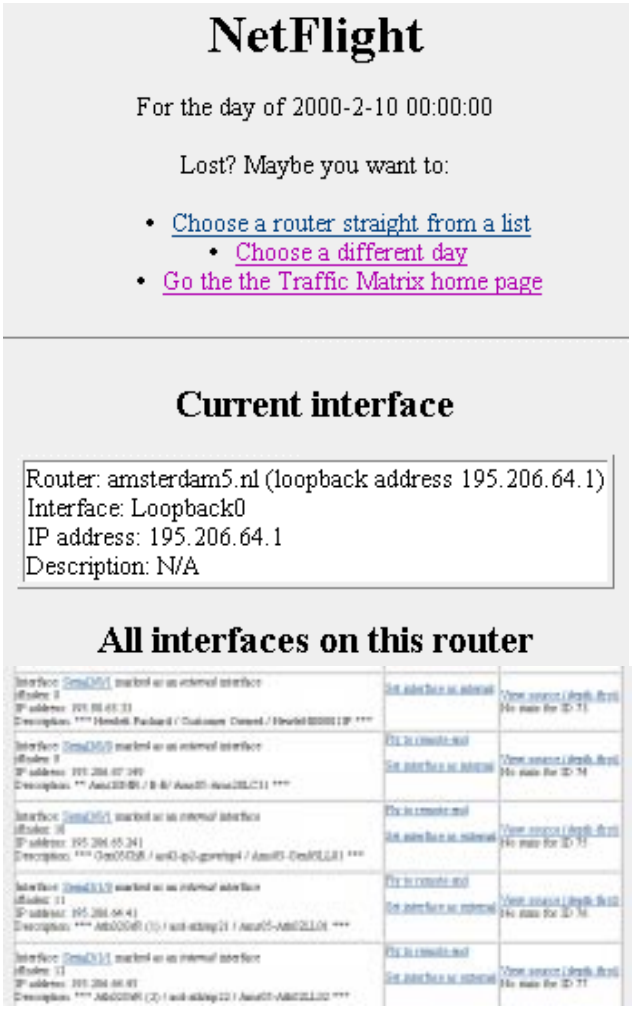
In order to understand the layout of the network it was necessary to have some way to visualise it. Our network topology existed in a number of forms, from Cisco router configurations, through to network operations applications (Spectrum). I investigated the possibility of interfacing the Traffic Matrix with Spectrum, however this proved to be a non-trivial exercise. Spectrum is also maintained manually, and I wanted to avoid any sort of manual input wherever possible in order to reduce the likelihood of errors. The Cisco router configurations were being pulled off the routers on a daily basis for management reasons, and it made sense to use these as a source of the network visualisation. No human input would be involved with the process and nothing can reflect the actual configuration of the network better than the router configurations themselves.

I wrote a parser to read in the router configurations and produce a network representation. A critical part involved calculating what interfaces of each router connected to what interface of every other router. I did this based on the IP address/netmask. The most efficient way to calculate the connections is O(nlogn), however for simplicity I am using an O(n^2) algorithm. The network has approximately 1000 interfaces, and so the number of comparisons required is quite large, but not unmanageable. In perl on a SPARC 1 this process takes about three minutes. The calculation is run as a cron process early in the morning and so the time taken really has no consequence.

Once all connections are calculated, the information is placed in a database. Visulisation of the resulting network is by way of cgi-bin scripts that interrogate the database in real-time. This results in a quick end-user response.

From the visualisation tool *Netflight*, it is possible to traverse the network and perform a number of valuable tasks. With it we can:

- Quickly pass from the interface of one router, to whatever router is connected at the remote end.
- Mark interfaces as being external or internal. This is necessary as only external interface statistics are used for the AS flow tables.
- View individual router interface statistics.



F. Active measurements

All measurements discussed so far have been passive. No actual test traffic has been placed on the network in order to measure its performance.

Of recent interest is the possibility of offering service level agreements to customers based on network delay and packet loss. It was decided to investigate this further as part of the Traffic Matrix project.

One way traffic delay measurements would have been preferred, however a suitable clock source is required on each collector ala the RIPE Test Traffic project. Roll-out of such equipment is non-trivial. Opting for an easy solution I went with round-trip delay measurements using *ping* with 64 byte packets.

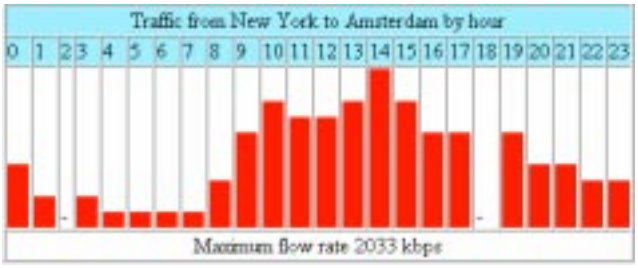
Each collector is configured to *ping* each other collector once a minute. The resulting timing information is transferred to the central server once a day. Once there it is imported into the database, and subsequent static pages in

matrix table format (sources against destinations) are generated.

G. Reporting

One of the main static reports generated each day is the Inter City traffic report (and its sibling, the Inter POP traffic report). These list sources of traffic down the side and destinations for the traffic along the top in matrix format. The average daily data flow rate between cities is shown in the relevant cell.

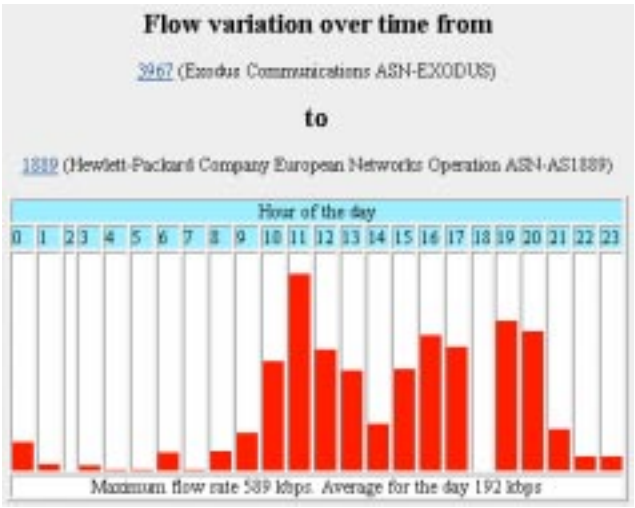
This matrix-style report acts as a starting point for network inquiries. From here, hyper-links lead off to other reports of interest. For each city to city cell, a link leads to a graph showing the traffic trend over the day.



This information is very useful for planning new network links. It shows the traffic the network is being asked to carry, and not necessarily the traffic flowing over a particular link. In the example graph given, we can see what traffic entering the network in New York, actually exits in Amsterdam. This is quite different to knowing how much traffic went over the New York to Amsterdam links.

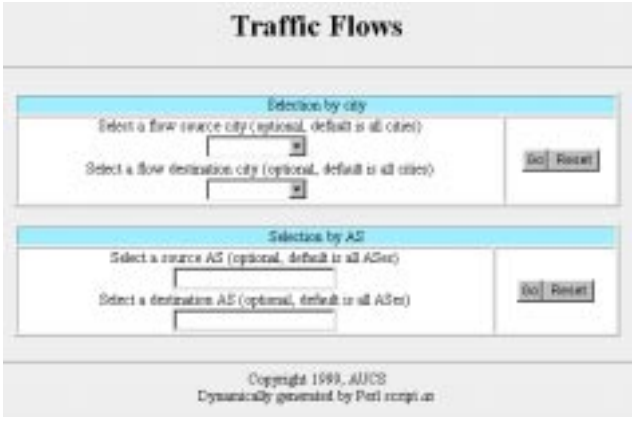
From the city to city matrix report it is also possible to view the AS to AS flows that make up the traffic between two cities. This makes it very easy to identify large traffic flows, and where they are going.

By drilling down further, we can look at an individual flow to see the source and sink of the data, as well as how it varies throughout the day. Also, information from the RIPE database is used to identify the holders of the AS numbers appearing in the report. Note that information in the RIPE database has tight restrictions on its use. Check out http://www.ripe.org for more information.



As a way of double-checking values, the raw statistics are also displayed. These are essentially the Netflow Export statistics after they have passed through aggregation to the level of one hour.

The links from the main city to city report actually point towards a dynamic cgi-bin script. It is also possible to call this script directly. In such cases the use is presented with a screen where they can enter source/destination cities of interest or source/destination ASes of interest.



The report generation for the active measurements gathered is similar in style to the city to city traffic flow report. Source POPs are listed down one side, with destination POPs along the top. For each POP combination minimum, average, maximum as well as standard deviation timing information is provided. An inter-POP reliability figure is also calculated based on the number of *ping* replies received.

III. SUMMARY AND NOTEWORTHY ITEMS

The Traffic Matrix as it stands today works well and provides useful reports for both Capacity Planning and Peering. There are however a number of shortcomings with the present system. These will be addressed in version 2 of the Traffic Matrix. This is presently under development.

- Traffic behind an AS – Of particular interest to peering is the possibility of identifying the traffic of a transit provider. Presently if we look at the traffic from a transit provider to our network, we see just traffic originating on that network, and not the transit they are carrying from other networks. By using the BGP routing table, it is possible to calculate this information. A public domain package called Sluice[3] is already available that works with Cflowd to provide this.

- Scalability for POPs – The O(n^2) traffic matrix displays do not work well for large numbers of sites. This is of particular interest to AUCS as the current pan-European[4] network is expanding to become a global network. This means the number of collectors in our network will more than double. By grouping collectors into *regions* I hope to be able to provide reports of manageable size. This is essentially the idea of going from a flat layout to a hierarchy with two layers.

- MySQL limitations – Present version of MySQL are limited to 4 GB table sizes. Our presently fastest growing table contains the AS to AS traffic flows. This table grows at a rate of about 30 MB per day. It's easy to see that after four months we will run into problems. An Informix (or equivalent) database may be a possible solution in this regard, having no such 4 GB limit.

- Database speed – A large amount of time is required each day to generate the city to city traffic flow matrix. For each city to city pair, 24 queries are run to produce the hourly trend graphs. With ten cities in the matrix, this equated to 2400 queries. MySQL is an efficient database, however running this many queries still takes a lot of time. As the size of the database increases, the query time also increases noticeably. Presently the query process appears to be disk bound, so a short-term solution may be to add more memory. A suitable long-term solution is not yet clear.

- RIPE one-way delays – Presently the RIPE Test Traffic project involves test boxes measuring one-way delay times between large Inter-networks. There are plans to make available for purchase this year, boxes for measuring one-way delays within a network. This is by far the best way to implement one-way measurements in our network. The output from the RIPE boxes could easily be imported into the same central server we are presently using.

[3] Sluice is available via ftp from ftp.false.net/pub/sluice and the author Gordon Mercer can be reached at gmercer@raver.net

[4] pan-European, plus presence in the US.