Proceedings of the 14th Systems Administration Conference (LISA 2000)

New Orleans, Louisiana, USA
December 3–8, 2000

Combining Cisco NetFlow Exports with Relational Database Technology for Usage Statistics, Intrusion Detection, and Network Forensics

John-Paul Navarro, Bill Nickless, & Linda Winkler – Argonne National Laboratory

ABSTRACT

Argonne National Laboratory operates a complex internal network with a large number of external network peerings. A requirement of this network is that it be monitored with minimal impact on traffic. Cisco NetFlow technology provides the information necessary to monitor such a network, but the data from NetFlow must be captured and analyzed. We present a system that uses a high-powered relational database to manage the data. Our primary motivations in building this system were to learn whether or not database technology was an appropriate tool for this situation and to understand what types of questions about the network could be answered with such a system.

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

The Problem

High Performance Network with Minimal Firewall

Argonne National Laboratory peers with more than 50 external Internet Service Providers and Internet2 networks, at rates up to OC-12 (622 million bits per second). Soon we expect those peerings to increase in speed to Gigabit Ethernet and higher. Argonne's networks are based on Asynchronous Transfer Mode (ATM) and switched 10/100/1000 Megabit/second Ethernet. Argonne's network has two separate border routers that handle these peerings. While these border routers can take over for each other in the case of a failure, normally they do not see each other's traffic.

This is an interesting situation for a number of reasons. First, there is no single point in our network that can be used to monitor all traffic into and out of the lab. Second, the external network speed is a critical issue to a number of research groups in the lab – slowing down the networks by running them through filtering or monitoring routers can have a huge impact on many experiments. And finally, the number of peerings and their exact topology frequently change as we modify our external networking relationships.

What's Going On in the Network?

We needed a way to examine network utilization statistics, perform basic intrusion detection, and look at traffic patterns. The obvious way to do this would

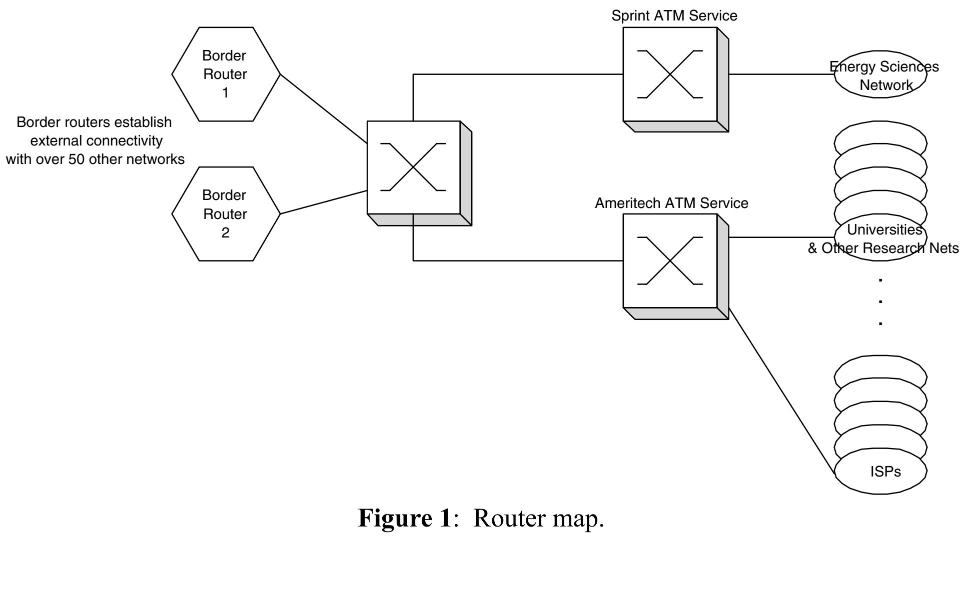
Border Router 1
Border Router 2
Border routers establish external connectivity with over 50 other networks
Sprint ATM Service
Ameritech ATM Service
Energy Sciences Network
Universities & Other Research Nets
ISPs

Figure 1: Router map.

be to drive all traffic through a router and use the router to watch every packet, but we are unwilling to create an artificial bottleneck in our network for network statistics gathering or intrusion detection purposes.

Unfortunately, many available network statistic gathering and intrusion detection systems, such as Network Flight Recorder, require that the intrusion detection system be able to inspect each packet as it enters and leaves the network. The Argonne network is not amenable to general packet inspection even at some small number of points for the reasons outlined above.

Thus we needed to find some way to keep an eye on our network traffic without measurably impacting its performance.

The Scenario

Fortunately, we had other options to explore.

Cisco has a technology known as "Netflow", which provides a summary of traffic through a router. We also have some significant experience with databases, which led us to believe that a database would be the optimal way of tracking Netflow data from multiple routers. Initially, we weren't sure how much data would be generated from our network, but, since we operate a number of supercomputers to support scientific experiments, we felt that we probably had the computing resources on hand to at least study the situation.

The motivating question for us was this: Is database technology a good way of tracking and analyzing NetFlow data?

Cisco Netflow Overview

Here is how Cisco describes this technology:

> A network flow is defined as a unidirectional sequence of packets between given source and destination endpoints. Network flows are highly granular; flow endpoints are identified both by IP address as well as by transport layer application port numbers. NetFlow also utilizes the IP Protocol type, Type of Service (ToS) and the input interface identifier to uniquely identify flows [1].

The IP routers in our network do NetFlow switching already, as it is a very efficient way of enforcing Access Control Lists (ACLs). The first packet of a flow will be inspected to see if it should be permitted or denied. Once that determination is made, a flow record is placed in the router forwarding cache. Following packets are matched against the flow record and are forwarded appropriately. Once the flow record cache line times out, a UDP/IP packet is generated and forwarded out of the router to a collection station.

The resulting NetFlow records provide a summary of just about anything a network statistics collector or intrusion detection system may want to know about the traffic, with the exception of the actual packet contents themselves.

Database Technology

We run a number of Oracle and MySQL database servers to support our researchers. One of the authors (Navarro) was a professional Database Administrator (DBA) for a large Oracle installation at a previous employer.

Over the past several years, we have moved many of our critical infrastructure applications such as our list of hosts, lists of users, and so on, from flat files or simple lists into relational databases. Thus, many of our staff are becoming experienced with using databases for systems and network administration.

Thus we were fairly confident that a database would be the right solution for storing NetFlow data and that we would have the ability to manipulate it as we wished. The remaining question was – what should we use for a database server?

High Powered Database Server

We support a collection of workstations, clusters, and supercomputers as well as our networks. Our existing database servers run primarily on small Linux boxes or Solaris machines, and we knew that none of them would be up to the task of keeping up with the NetFlow data. The only spare computers we had were obviously underpowered.

One of the supercomputers that we support is a 96-processor SGI Origin 2000 [2] with approximately 2 terabytes of Fibre Channel disk. This machine is primarily used to support computer science and computational science. So, we decided to run this project as an experiment and, therefore, to justifiably use the Origin 2000 to run the database. We were fairly confident that it would have sufficient computing power to keep up with the NetFlow data during our initial tests, after which we could more accurately predict what kind of system would be necessary for production.

The Creation

Here, we describe the system that we built to capture and analyze the router NetFlow data.

Netflow to Database on Origin 2000

There are three basic parts to our configuration: the actual routers generating NetFlow data, a Perl [3] script to catch the NetFlow data from the network, and the back-end SQL database running on the Origin 2000. We experimented with Oracle 8i [4] and MySQL [5] back-end SQL software. We used on-line data capture where incoming NetFlow data went into the database directly. Our Perl script that caught the NetFlow data from the network would simply make DBI [6] calls to insert the data into the back-end SQL database. The primary advantage of this method is that the database is updated in real time, but has the disadvantage that data can be lost if the back-end SQL database is unavailable for any reason.

We also used off-line data capture where the incoming NetFlow data was written to a temporary disk text file and then periodically loaded in bulk to the back-end SQL database. This allowed us to continue capturing data while the back-end SQL database was unavailable. It also allowed us to insert the data into multiple back-end SQL database engines when we wanted to compare their performance.

Database Schema Design

We chose a very simple database schema, drawn directly from the NetFlow specification itself. Our database consists primarily of a single table. Each row in the table is a single NetFlow record. Each column in the table has a one-to-one correspondence with NetFlow fields. We added only one extra column in the table to identify which border router generated the flow record.

```
create table netflows (
   router_id  char(1)            not null,
   src_ipn    bigint    unsigned not null,
   dst_ipn    bigint    unsigned not null,
   nxt_ipn    bigint    unsigned not null,
   ifin       smallint  unsigned not null,
   ifout      smallint  unsigned not null,
   packets    integer   unsigned not null,
   octets     integer   unsigned not null,
   starttime  timestamp          not null,
   endtime    timestamp          not null,
   srcport    smallint  unsigned not null,
   dstport    smallint  unsigned not null,

   tcp        tinyint   unsigned not null,
   prot       tinyint   unsigned not null,
   tos        tinyint   unsigned not null,
   srcas      smallint  unsigned not null,
   dstas      smallint  unsigned not null,
   srcmask    tinyint   unsigned not null,
   dstmask    tinyint   unsigned not null
)
```

This schema is so simple because we did not want to hinder our ability to make queries across the data later. We were interested to see how far modern SQL database engines could take us without optimizing the tables for particular types of queries.

Using The System

The system as described above is actually the complete system that we are using at present. We created the infrastructure to get the data from the routers into the database and then left it at that. Future steps (as discussed below) might be to provide nice interfaces to the data in the database, but this wasn't our initial goal. Rather, we wanted to see what kinds of questions could be answered with the data. From this point on, we've experimented with the system by forming basic SQL queries and examining the results. Our simple database scheme was designed with exactly this usage in mind.

Once you have the NetFlow records stored in the relational database, you can run many different types of queries against them. Here, we present some of the ad-hoc queries that we've found that we frequently use.

Usage Statistics

This is an example SQL query for someone interested in network statistics, perhaps asking the question "what Autonomous Systems (ASes) have we exchanged the most amount of traffic with recently?"

```
# Give a list of interesting
# ASes (high traffic flow)
select srcas,dstas,sum(octets) as bytes
       from netflows
       group by srcas,dstas
       having bytes > 10000000
       order by bytes
```

The resultant table lists each pair of Source AS and Destination AS that we have exchanged traffic with,

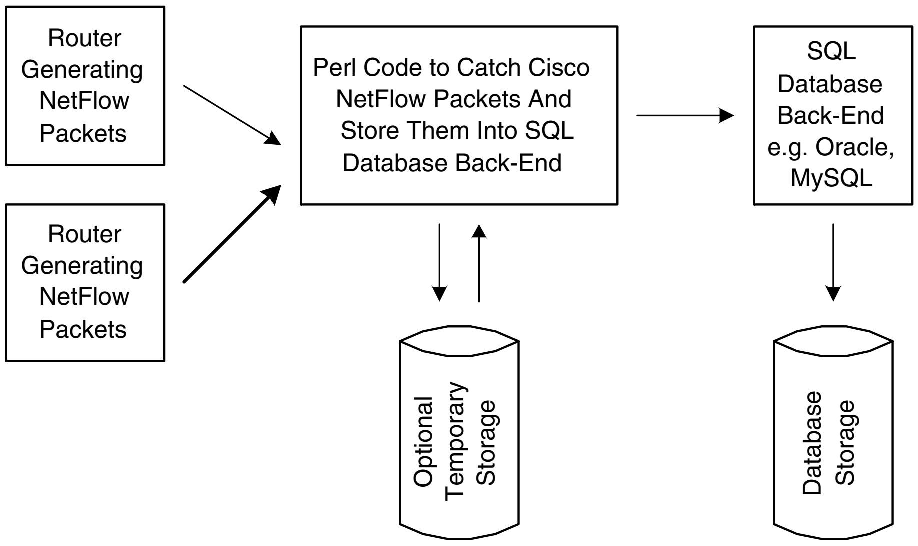
Router Generating NetFlow Packets
Router Generating NetFlow Packets
Perl Code to Catch Cisco NetFlow Packets And Store Them Into SQL Database Back-End
SQL Database Back-End e.g. Oracle, MySQL
Optional Temporary Storage
Database Storage

Figure 2: Database layout.

including our own, with more than 10 million bytes throughout the data stored in the NetFlows table, with the Source AS/Destination AS pairs that exchange the most data at the top.

Intrusion Detection

A network security officer might be interested in what outside IP addresses are trying to perform scans. Here is a query that illuminates the answer to that question:

```
select src_ipn,count(distinct dst_ipn)
                     as num_anl_addrs
   from netflows
   where srcas>0 and starttime >
         date_sub(now(),interval 3 day)
   group by src_ipn
   having num_anl_addrs > 64
   order by num_anl_addrs
```

This query gives the network security officer a list of IP addresses from external ASes that have contacted a large number of internal IP addresses. Given an IP address high on this list, the security officer might want to see what that IP address has been doing:

```
select src_ipn,min(starttime) as first,
          max(endtime) as last,
          count(distinct dst_ipn) as
                   addrs,prot,dstport
       from netflows
       where src_ipn=140221009006
       group by prot,dstport
       order by addrs
```

ICMP Echo Reply (ping) scans, port scans, and even NMAP stealth scans show up very obviously in this and the preceding report.

Network Forensics

Let's say that the network security officer suspects that a machine on her network has been compromised. The officer would like to go back in time to see where any network connections came from, what kind of network protocols were used to compromise the host, and see where any outgoing connections may have gone. The officer may even wish to look at other machines on the same subnet. Here is one query that might be used to do this; see Listing 1. The officer then gets a list of all the traffic entering or leaving a network between two given points in time. Such a report can be used to narrow down the type of attack that might have been used, and whether the compromised machine(s) on that network were used to attack hosts elsewhere.

Results and Practical Applications

Sizing It: Performance and Resource Impact

Not everyone has a spare 96-processor Origin 2000 sitting around to host a NetFlow database. (Not even us, really.) Thus, one has to ask questions about the investment necessary in time, computer hardware, and software to appropriately host a NetFlow database.

The first variable to consider is the rate of NetFlow records generated by your routers. This rate will vary widely depending on your usage, whether your network is being actively scanned, and the time of day. The best way to get a handle on this question is to simply go ahead and install the Perl code that catches NetFlow packets and saves them as flat files. Run this for a few days, preferably during the workweek as well as over a weekend. You can then run simple line counting utilities over these flat files.

Our routers generated about 14 million flow records on Monday, July 24 2000. Our Perl script stores the flow records in 5-minute batches. These batches ranged from 26,564 to 80,634 records per batch.

It is important that your database engine can accept table inserts much faster than your routers can generate NetFlow records. You need some performance overhead to handle queries, outages, and timeouts. We recommend trying to choose a SQL back-end system that can handle at least triple your real-time NetFlow record generation.

Using our example from July 24, we would need a SQL back-end database that can handle about 1500

```
# Something very weird happening at a particular time on a particular net.
select min(starttime) as startt, max(endtime) as endt,
       src_ipn, dst_ipn, prot, srcport, dstport,
       sum(packets) as pkts
  from netflows
 where ( ( starttime > 20000105003200
           and starttime < 20000105003600 )
      or ( endtime   > 20000105003200
           and endtime   < 20000105003600 ) )
 group by src_ipn,dst_ipn,prot,dstport
 having ( src_ipn >= 146137000000
          and src_ipn <= 146137255255 ) or
        ( dst_ipn >= 146137000000 and
          dst_ipn <= 146137255255 )
 order by pkts,src_ipn,dst_ipn,prot,srcport,dstport
```

Listing 1: Examining other machines on the same subnet.

insertions per second at a bare minimum. Preferably we would want a database that could handle triple that rate, or just under 5000 insertions per second.

The next variable to consider is how long you want to keep NetFlow records around. This dictates your overall storage requirements. Be aware that the size of your NetFlow table may impact the speed of table inserts, which feeds back to into the performance requirement discussed above.

We chose to keep about two weeks' worth of data on line. This time frame is a compromise between our desire to protect the privacy of our network users and having the data around to look into unusual events. To protect the long-term privacy of our network users we specifically chose not to back up the database itself.

Turnaround Time vs. Insertion Performance

You will also choose which table columns to index, if any. This is a tradeoff between insert performance and query performance. If you intend to simply capture data and look at it when something bad happens, you may not wish to spend the cycles indexing data that will be thrown away. The downside is that every query will require a full scan through the data before it can return results.

Some queries just about require a full table scan, such as the Intrusion Detection example query we provide above. If you anticipate doing those types of queries on a regular basis, you might choose to buy the fastest disk system you can find and not bother doing any indexing at all.

Database Technology Choice Makes a Big Difference

As mentioned above, we tried MySQL and Oracle on the SQL database back-end host. As part of their license agreement, Oracle doesn't let you publish specific performance numbers. But we can make some general comparisons between the two systems.

Oracle costs money. MySQL is (pretty much) free.

Oracle supports transactions, which slows it down but protects the database against corruption if the underlying host crashes for some reason. If the host system crashes under MySQL, you often have to re-index the table completely, which can take a very long time.

The MySQL query optimizer is sometimes less sophisticated than we might wish. Consider the query in Listing 2, assuming that the src_ipn and dst_ipn columns are indexed. This query took about 8 minutes to run, returning 576 records. The MySQL engine did a full 175,460,008-record table scan looking for matching rows.

However, the slightly simpler query in Listing 3 took 0.04 seconds to return 16 records. This time the MySQL engine used the src_ipn index to find the matching records much more efficiently than doing a full table scan. (The other obvious query using the dst_ipn index took about five seconds and returned the other 560 rows.) A more sophisticated query optimizer than MySQL provides would have split the original query into the two much simpler-and faster-queries and then combined the results.

If you've already decided to use MySQL, obviously you will want to run the two fast queries instead of the one slow query. But if you haven't yet decided on a SQL database back-end system, you may wish to run some sample queries and see how the optimizer treats each of them. The query optimizer is also critical to achieving good performance in a SQL database that supports parallel queries.

Other database products may provide features that could be used to speed up certain queries. Consider the Usage Statistics example query. If one were to drop the time constraint, it might be a perfect fit for an IBM DB2 [7] Summary Table. That is, the statistics would be generated "on the fly" during the NetFlow record insert process.

```
select src_ipn,dst_ipn,prot,srcport,dstport,
            sum(octets) as bytes, min(starttime) as first,
            max(endtime) as last
       from netflows
       where src_ipn=146139112126 or dst_ipn=146139112126
       group by src_ipn,dst_ipn,prot,srcport,dstport
       order by bytes
```

Listing 2: One form of query.

```
select src_ipn,dst_ipn,prot,srcport,dstport,
            sum(octets) as bytes, min(starttime) as first,
            max(endtime) as last
       from netflows
       where src_ipn=146139112126
       group by src_ipn,dst_ipn,prot,srcport,dstport
       order by bytes
```

Listing 3: Quicker query.

Futures and Conclusions

Non-Supercomputer Database Hosts

We cannot dedicate our 96-processor Origin 2000 to this application. So we must choose the most efficient combination of disk drive(s), CPU, memory, and database software to meet our NetFlow record generation rate and query types.

We may even choose to run several different SQL database back end servers, each optimized for a particular type of query.

We are still in the process of deciding exactly what system to use for this. As usual, it comes down to more of a question of budget than necessary technology.

Higher Level Tools Necessary For Non-DBAs

We are fortunate to be a cross-disciplinary team of network professionals, system administrators, and database administrators. Each of us is comfortable with writing simple ad-hoc SQL queries to look at the NetFlow data. Over time we hope to build higher-level tools that could be accessible to a wider audience. Examples might include a web interface that network security officers could use to view the activities of a given host, email alerts based on intrusion detection queries, or even graphics of network utilization over time.

Database Technology Is a Good Solution

The good news is that the higher-level tools can be built using the wealth of tools available for SQL databases. There is any number of commercial and/or open-source systems available to make web pages that use SQL database back ends. Just about any report-writing program can take data from a SQL query.

Database technology provides a good abstraction for this problem. It separates the nitty-gritty details of data storage from the actual problem we are interested in solving. As database software technology improves we get the benefits of those improvements right away. We are not stuck with a particular architecture tuning that limits us from following the Moore's Law increases in processor and disk storage performance. As our needs increase, we can swap out an under-performing SQL back end system entirely, as needs and budgets permit. Finally, we can take advantage of all the data protection and administrative expertise of an existing SQL infrastructure.

Availability

All of the code we used in this work is written as Perl and Bourne Shell scripts. There are probably no more than about 200 lines of actual source code. Look for a pointer to the NetFlow Database Perl Code on the http://www.mcs.anl.gov/systems/software/ web page by the time this paper is presented at LISA.

Author Information

John-Paul Navarro <navarro@mcs.anl.gov> has been working at Argonne National Laboratory for 3 years. His professional interests include Linux clusters, scalable cluster management, and relational databases.

Bill Nickless <nickless@mcs.anl.gov> is a 9-year veteran of Argonne National Laboratory. His professional interests include high performance networking and data storage. He is never more happy than when fine tuning BGP policies while putting bar code labels on magnetic tape cartridges.

Linda Winkler <winkler@mcs.anl.gov> is Senior Network Engineer at Argonne National Laboratory's Mathematics and Computer Science Division. She also serves as MREN Technical Director and is a member of the STARTAP engineering team. Her focus since 1995 has been in the area of interconnectivity and interoperability of wide-area research networks in support of advanced scientific and engineering applications.

References

[1] http://www.cisco.com/warp/public/cc/cisco/mkt/ios/netflow/tech/napps_wp.htm .
[2] http://www.sgi.com/origin/ .
[3] http://www.perl.org .
[4] http://www.oracle.com/database/oracle8i/ .
[5] http://web.mysql.com/ .
[6] http://search.cpan.org/doc/TIMB/DBI-1.13/DBI.pm .
[7] http://www.ibm.com/db2 .