sFlow

I can feel your traffic

Elisa Jasinska
Amsterdam Internet Exchange
elisa.jasinska@ams-ix.net

Abstract

The explosion of internet traffic is leading to higher bandwidths and an increased need for high speed networks. To analyze and optimize such networks an efficient monitoring system is required.

An Internet Exchange (IX) interconnects various network providers, for example ISP's. The Amsterdam Internet Exchange (AMS-IX) is by its amount of traffic the biggest Internet Exchange in the world. To give the AMS-IX members more insight into their peering traffic and provide information to optimize the network structure, AMS-IX is using sFlow for its traffic analysis.

A throughput average of more than 120 Gb/s gets analyzed by an open source software developed in PERL.

1 Introduction

The widely-used NetFlow is a Cisco IOS software feature wherein a Cisco router exports flow-records to software running on a server, which can further analyze the traffic.

Collecting the NetFlow data can be expensive for the router's CPU because every packet has to be "touched" in order to get analyzed. Later on Cisco introduced "Sampled NetFlow" where the router is only looking at every n-th packet to maintain the NetFlow records. This way might be more efficient, but Cisco doesn't provide any additional information like how the packets are sampled (for example randomness) and there are no existing standards or RFC's describing the sampling mechanism.

Another way to get sampled flow data is sFlow. It's a relatively new standard for monitoring high speed networks described in RFC 3176. A solution to collect, analyze and store sFlow data has been implemented on the platform of the Amsterdam Internet Exchange [1], which will be further discussed in this paper.

The AMS-IX and the goals to achieve by using sFlow are described in Section 2 of this paper, Section 3 gives an overview about the sFlow sampling mechanism. More information about available collector software can be found in Section 4. The software written for AMS-IX (sFlux) is shown in Section 5 and results are shown in Section 7, some benchmarks are described in Section 8 and we conclude in Section 9.

2 AMS-IX

The Amsterdam Internet Exchange is a non-profit Internet Exchange based in Amsterdam. Over 250 parties from all around the world exchange IP traffic across the AMS-IX platform. Internet Service Providers, international carriers, mobile operators, content providers, VoIP providers, application providers, web hosters and other related businesses are all peering with each other across the AMS-IX platform [1].

At four independent co-location facilities in Amsterdam, 253 members are currently[1] connected to 420 ports at AMS-IX. Each member can choose one or more co-locations to set up their hardware and connect to one of the AMS-IX ethernet switches there [2], or choose to connect via a so-called "pseudowire" to one of the AMS-IX Ethernet switches via a third party.

2.1 Traffic

The Amsterdam Internet Exchange is the biggest IX in the world with a traffic average of 136 Gb/s and peaks up to 208 Gb/s (see Figure 1) [3].

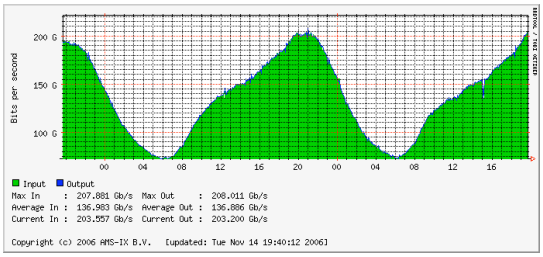

Figure 1: Current AMS-IX traffic statistics

In May 2005, the traffic volume (in/out) was 15913/15905 TB per month. A year later (May 2006) it raised up to 32458/32432 TB/month. On average, traffic volumes doubles every 12 months.

2.2 AMS-IX Requirements

By using sFlow AMS-IX would like to provide more information to its members. On the basis of the traffic graphs each member can see the amount of traffic on their interfaces. These graphs are generated from the interface counters for each port, which are polled via SNMP from the switches. But the members don't know to which other member the traffic is going. As sFlow samples provide the source and destination MAC addresses, and AMS-IX only allows one MAC address per member port, one can trace statistically the amount of traffic going from one member to another.

[1] November 16, 2006

Besides IPv4, IPv6 and ARP AMS-IX doesn't allow any ether types on the platform. But how much ARP and IPv6 traffic is there actually sent across the network? By analyzing the captured ethernet header one can compute a percentage of each ether type on the network.

Due to the high traffic rate on the AMS-IX platform the solution has to deal with a huge amount of data which needs to be processed. Especially the performance while saving and displaying the information must be acceptable.

In order to visualize the data, it has to be parsed, analyzed and stored. Some of the existing sFlow tools interact with databases like MySQL or Postgres (more on that in Section 4), but the data still needs to be processed for visualization. A common way to generate graphs is RRDTool [13], which provides a Round Robin Database to store the required information for graphical output. It seems obvious that in this case databases like MySQL or Postgres could be skipped, as there is no need to store more data than needed for the visualization. In case of common databases one also needs to take care of the huge amounts of old stored data, which is done automatically in a Round Robin Database.

3 sFlow

The sFlow standard describes a mechanism to capture traffic data in switched or routed networks. It uses a sampling technology to collect statistics from the device and is for this reason applicable to high speed networks (at gigabit speeds or higher) [4].

- An *sFlow agent* is the implementation of the sampling mechanism on the hardware (for example a switch).
- The *sFlow collector* is a central server which collects the sFlow datagrams from all agents to store or (later) analyze them.

In the remainder of this section we explain the different sampling methods and the sFlow protocol.

3.1 Sampling Methods

The *sFlow agent* uses two forms of operation: (i) time-based sampling of counters, and (ii) statistical packet-based sampling of switched or routed packets.

Counter Sampling

A *polling interval* defines how often the sFlow octet and packet counters for a specific port are sent to the collector, an sFlow agent is free to schedule polling internally in order maximize internal efficiency.

Packet Based Sampling

Based on a defined *sampling rate*, either for the complete agent or for a single interface, one out of N packets is captured. This type of sampling does not provide a 100% accurate result but it does provide a result with acceptable accuracy [5].

Packet Based Sampling Example

If 1,000,000 packets are transmitted through a network and random samples of 0.25% (sampling rate: one out of 400) are taken, 2,500 packets will be captured. If 100 of these samples are IPv6 traffic, the minimal amount of IPv6 packets must be 100, because we have seen them. The maximal amount could be 997,600, because we have 997,500 unseen packets and we know that 100 are IPv6 traffic. However, neither of these two values is at all likely. Most likely is that the ratio for all the packets is nearly the same as in the sampled packets.

In general, the accuracy depends on the number of samples which is very useful in high speed networks. The same sampling rate produces much more samples in a network with a high throughput than in one with a low throughput. For example on a switch with traffic averaging at 400 frames/second and a sampling rate of one out of 400, we get 1 sample/second. On a switch with a traffic of 40,000 frames/second we can get the same result by capturing only one out of 40,000 frames.

When the sampling intervals are short or the traffic rates low, the results can be very inaccurate. If we have only one IPv6 packet per second which has been monitored for an hour we will have 3.600 IPv6 packets in that hour. With a sampling rate of one out of 400 we will see only around 9 sampled IPv6 packets in that given hour.

In order to improve the accuracy we could increase the sampling rate or the time of sampling. Increasing the sampling rate also means more computing for the switch and could result in high CPU loads. Extending the timeframe means that the samples are analyzed over days, weeks or month, which does not always satisfy the requirements. For applications, like billing systems, which require a larger accuracy, extending the timeframe is fine but for real-time analysis it is not [5].

3.2 sFlow Datagram

The sampled data is sent as a UDP packet to the specified host and port on the *sFlow collector*. The default port is 6343.

The lack of reliability in the UDP transport mechanism does not significantly affect the accuracy of the measurements obtained from an sFlow agent. If counter samples are lost then new values will be sent when the next polling interval has passed. The loss of packet flow samples is a slight reduction in the effective sampling rate. The use of UDP reduces the amount of memory required to buffer data. UDP is more robust than a reliable transport mechanism because under overload the only effect on overall system performance is a slight increase in transmission delay and a greater number of lost packets, neither of which has a significant effect on an sFlow-based monitoring system.
The UDP payload contains the sFlow datagram[2]. Each datagram provides information about the sFlow version, its originating agent's IP address, a sequence number, how many samples it contains and usually up to 10 *flow samples* or *counter samples*, see Figure 2.

[2] These examples are based on sFlow version 4

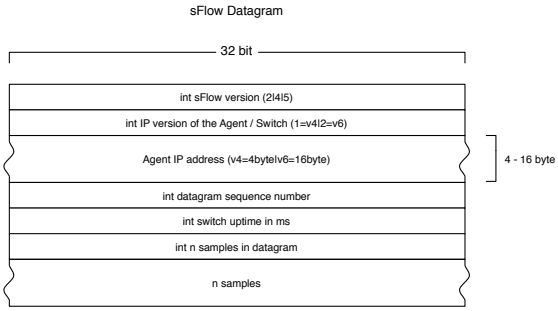

Figure 2: sFlow Datagram Structure

A *flow sample* consist of *packet data* and *extended data*. The *packet data* will typically contain a *sampled header structure*, which is the whole sampled packet up to 256 byte. If the agent is incapable of taking a sample of the whole packet, there are also *sampled IPv4* and *sampled IPv6 structures* defined. These contain only the IP header data of the sampled packet. Each sample provides the input and output interface as well as the sampling rate for the given port, see Figure 3. The *extended data* structure provides additional information, for example in case of a switch the source and destination VLAN.

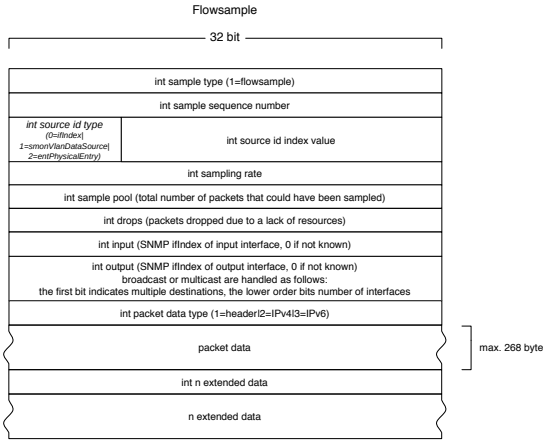

Figure 3: sFlow Flowsample Structure

RFC 3176 [6] describes the sFlow version 4 datagram structure.

v.5

The InMon[3] memo [7] describes sFlow version 5. Records within the datagram now have tag, length and value information, in order to have the possibility to skip unknown types. A global name space is defined for record types, allowing implementors to add their own record types. Extensibility also permits the addition of new record types without requiring changes to the published sFlow protocol [7].

In order to get a better understanding of the data structure, we created data format diagrams for sFlow version 4 [8] and sFlow version 5 [9] which are by now also available on the sFlow specification website[4].

4 Existing sFlow software

There are a couple of solutions to collect and decode packets with sFlow data, they are listed on sFlow.org [10]. The following subsections illustrate the drawbacks subsidary for all of those tools.

4.1 sflowtool

InMon provides an sFlow toolkit to decode sFlow data. The current version of the core component sflowtool is 3.10. This is an easy to use and scriptable command line tool [11]. While listening to the specified port it will print the decoded data to STDOUT, which could be piped into another script for further analysis. Nevertheless the overhead in extra parsing would be to much considered the amount of traffic on the AMS-IX network.

4.2 pmacct

Pmacct is a suite of daemons for traffic monitoring. “sfacct” is the daemon to decode sFlow datagrams. The pmacct package is under strong

[3] InMon Corp. is focused on the development of traffic monitoring solutions for high-speed switched networks

[4] http://www.sflow.org/developers/specifications.php

development and the currently available version is 0.11.1. It provides plugins to store the collected data into databases like MySQL or PostgreSQL [12]. An example command line follows:

```
$ sfacctd -c src_mac,dst_mac -P print -r 10
```

This would print the source host, the destination host and the protocol to STDOUT. The option "-P mysql" would save the received data to a MySQL database and the option "-P pgsql" to PostgreSQL. But storing each sample in a database considered the amount of samples was not an option for AMS-IX.

4.3 Summary

Unfortunately neither of the existing software solutions is in line with our wishes. Each of the existing tools must be supplemented with additional scripts to create the desired output. This would be too expensive in terms of performance. So we decided to implement our own sFlow solution, called **sFlux**, which is adjusted to our needs.

5 sFlux

sFlux is a software package written in PERL for collecting and analyzing sFlow datagrams. sFlux.pl is built around Net::sFlow, an sFlow decoding module, which we open sourced on CPAN [14].

5.1 sFlux.pl

The daemon sFlux.pl receives UDP datagrams and analyzes the decoded information. It gets cached and is passed to RRDtool [13], to store and visualize it, in 5 minute intervals.

5.2 Net::sFlow

The module Net::sFlow decodes all currently available sFlow versions. The UDP payload of the sFlow packet is simply passed to the decode function of Net::sFlow [14].

```
($datagram, $samples, $error) =
    Net::sFlow::decode($udpObj->{data});
```

The function decode() returns a HASH reference containing the datagram data, an ARRAY reference with the sample data (each array element contains a HASH reference for one sample) and in case of an error a reference to an ARRAY containing the error messages. All provided HASH keys can be found in the documentation [14].

5.3 sFluxDebug.pl

The perl script sFluxDebug.pl can be used to display the received data to STDOUT.

```
$ ./sFluxDebug.pl
Port: 6343
Listening...
```

When receiving a packet the whole returned data structure will be printed to STDOUT.

5.4 Deployment Feasibility

All of the ca. 400 AMS-IX member-ports could possibly talk to each other which makes a total of 160,000 conversations. Currently we are writing around 40,000 RRD files every 5 minutes in 8 seconds. Tests have shown that writing up to 130,000 files is possible within 27 seconds on the given hardware[5].

5.5 Graphing

Creating the graphs doesn't need to be done after receiving a datagram, as data is inserted into RRD files only once every 5 minutes. Therefore, it's possible to create graphs on a periodic basis or on demand with separate scripts.

6 Results

The main goals for AMS-IX were to obtain an overview of the proportion of various ether types passed across the shared medium and to provide member-to-member traffic information.

[5] 2 CPU's (AMD Opteron(tm) Processor) each 2 GHz and 4 GBytes memory

6.1 Ether Type

Each sample is analyzed for the ether types in the ethernet header. As AMS-IX doesn't allow any ether types besides IPv4, IPv6 and ARP, the rest is summed up as "other". The amount of different ether types in every datagram is counted and a percentage is computed. The averages of the percentages calculated during a timeframe of 5 minutes are stored in an RRD file.

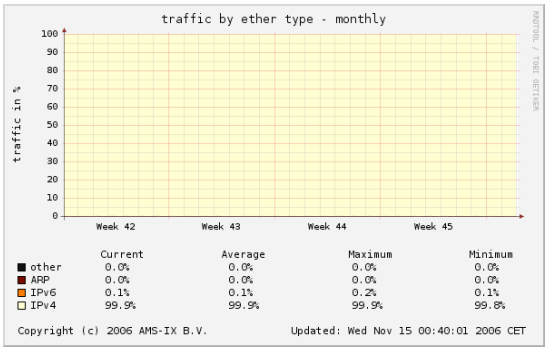

Figure 4: Ether Type Monthly

Figure 4 shows that on average 99.9% of the traffic is IPv4. Even if 0.1% does not seem a lot, considered that the total amount of traffic on the AMS-IX platform it is up to 200Gb/s, it is not insignificant.

6.2 Total IPv6

While already looking at the ether type, we sum up the data length of the IPv6 packets to get a total amount of IPv6 traffic in bits per second as shown in Figure 5.

6.3 Member to Member

The source and destination MAC address for each sample is analyzed. To calculate the amount of traffic in bits, the data length from the packet[6] is multiplied by the sampling rate and by 8 (because the data length is given in octets). This is summed up over the given time interval and divided by seconds to get a "per second" bit rate.

[6] Which includes the ethernet and IP header length

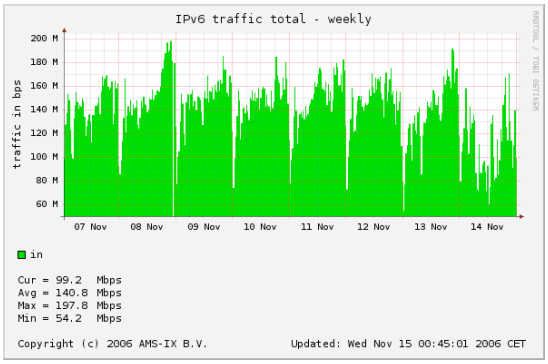

Figure 5: Total IPv6 Traffic

```
while (*inside a 5 min interval*){
    $data =
        $sFlowSample->{HeaderDatalen} *
        $sFlowSample->{samplingRate};
    $datasum += $data;
}
$bytePerSec = $datasum / 300;
```

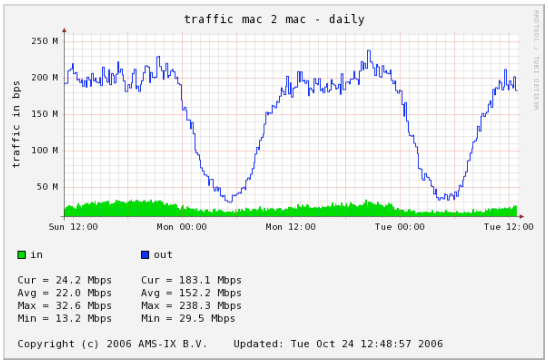

Figure 6: Daily Member-2-Member Graph

Figure 6 shows an example of a member to member graph. The area shows the traffic going from the given member to one other, the line shows the traffic coming from the other member to the current one.

7 Hardware Benchmarks

In the AMS-IX Lab, the available Foundry Network switches [16] and a server to collect the sFlow data were setup to test the performance. During testing the traffic was generated by an

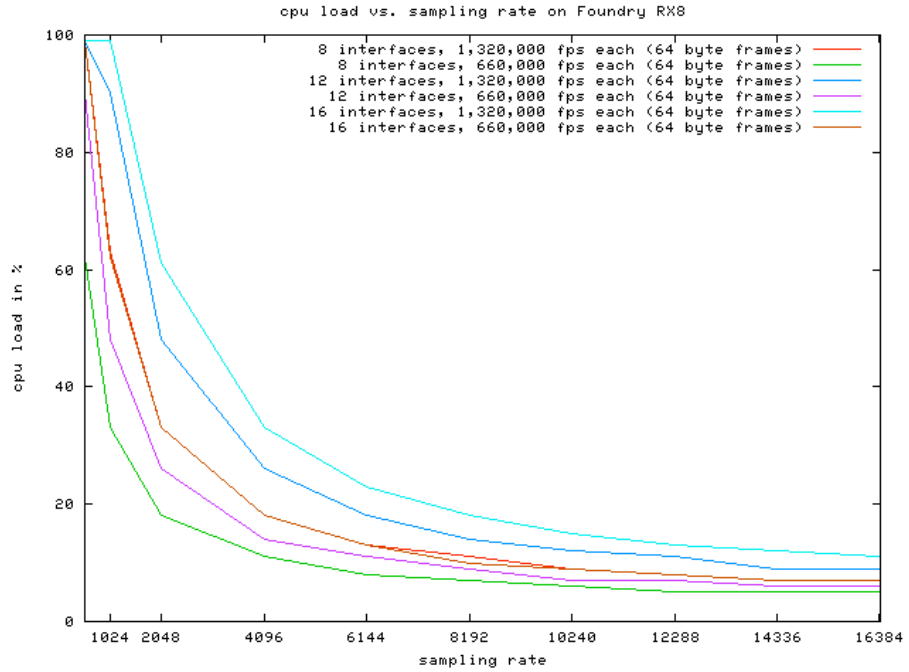

Figure 7: CPU load vs. sampling rate on Foundry RX8

Anritsu MD1230B - IP / Ethernet / POS Quality Analyser [15].

The hardware benchmarks were focused on CPU load of the switches with different sampling rates and throughput. The actual amount of samples taken within a second, depends on the per second frame rate as there is always one out of N samples taken.

7.1 Foundry BigIron 15000

The Foundry BigIron JetCore series switches compute the sFlow samples through an ASIC (application specific integrated circuit) so the switch CPU is not used to collect sFlow samples. This allows a reasonably high sampling rate, but the software IP stack packs the samples into the UDP packets, so with a high frame rate the highest configurable sampling rate (one out of 2) does not work anymore.

7.2 Foundry RX Series

The Foundry RX series computes the sFlow samples through the blade CPU. It depends highly on the defined sampling rate and the throughput how much CPU is used.

On sixteen Gigabit Ethernet interfaces with each ca. 1,320,000 fps as input, it produces a total of 21,120,000 fps to be sampled. With a sampling rate of 8192, ca. 2578 samples will be taken each second, using 18% of the blade CPU. If we decrease the frame rate by using only 8 ports, 11% CPU is used. If we decrease the frame rate per port to 660,000 fps, still with 16 ports, the CPU load will be 13% (see Figure 7).

8 Conclusion

The main bottleneck with the existing tools is to write all the received information to disk instead of focusing on the needed information. Common databases are not applicable with the

amount of traffic on the AMS-IX platform.

sFlux and the module Net::sFlow have been profiled very carefully to make it as efficient as possible. Our tests show that it is feasible to use even in a high throughput environment like the AMS-IX.

9 Acknowledgments

This paper was prepared for the 23rd Chaos Communication Congress, December 2006 in Berlin. It is based on my, at the time not yet published, student research project at the Amsterdam Internet Exchange supervised by Fabian Schneider, Technical University Berlin, Research Group Prof. Anja Feldmann. Once published[7], the elaborate paper will provide more detailed information and various test results.

I would like to thank the Amsterdam Internet Exchange, especially Henk Steenman, Ariën Vijn, Niels Bakker, Geert Nijpels and all the other NOC engineers, for the opportunity to work there and all the help I got during this project.

Also I would like to thank Tobias Engel for his help and support.

References

[1] Amsterdam Internet Exchange http://www.ams-ix.net, 2006.

[2] Amsterdam Internet Exchange - Technical http://www.ams-ix.net/technical/, 2006.

[3] Amsterdam Internet Exchange - Traffic http://www.ams-ix.net/technical/stats/, 2006.

[4] Informations, specifications, latest developments, and products about sFlow. http://www.sflow.org, 2006.

[5] Packet Sampling Basics http://www.sflow.org/packetSamplingBasics/index.htm, 2006.

[6] sFlow RFC 3176 http://www.ietf.org/rfc/rfc3176.txt, September 2001.

[7] sFlow Version 5 Memo http://sflow.org/sflow_version_5.txt, July 2004.

[8] sFlow v.4 Data Format Diagram http://jasinska.de/sFlow/sFlowV4FormatDiagram/, 2006.

[9] sFlow v.5 Data Format Diagram http://jasinska.de/sFlow/sFlowV5FormatDiagram/, 2006.

[10] sFlow Collector List http://sflow.org/products/collectors.php, 2006.

[11] InMon - sFlow Tools http://inmon.com/technology/sflowTools.php, 2006.

[12] Promiscuous mode IP Accounting package http://www.pmacct.net, 2006.

[13] Logging & Graphing with RRDtool http://oss.oetiker.ch/rrdtool/, 2006.

[14] Net::sFlow - PERL module decoding sFlow data http://search.cpan.org/~elisa/, 2006.

[15] Anritsu MD1230B - IP / Ethernet / POS Quality Analyser http://www.eu.anritsu.com/products/default.php?p=189&model=MD1230B, 2006.

[16] Foundry Networks BigIron http://www.foundrynet.com/products/family/bigiron.html, 2006.

[7] It will be available here: http://jasinska.de/sFlow