ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Andrea Weisskopf

Plug-ins for DDoS Attack Detection in Realtime

Semester Thesis SA-2004.19
November 2003 to May 2004

Tutor: Thomas Dübendorfer
Co-Tutor: Arno Wagner
Supervisor: Bernhard Plattner

Abstract

With the increasing threat of DDoS attacks to the Internet, tools to recognise them and initiate countermeasures become more and more important.

We try to help a network operator and provide visualisation plug-ins for UPFrame to get an impression in real-time what is going on in the network. Our traffic analysis is focused on exported Netflow streams from border gateway routers. Implemented are a flow size histogram plot and an IP activity map.

Contents

List of Figures

Chapter 1

Introduction

In this chapter we will first give a motivation for our work, present the task description and conclude with examples of other research projects pointing in the same direction.

1.1 Motivation

DDoS in general Distributed Denial of Service (DDoS) attacks are a threat to Internet services ever since the widely published attacks on ebay.com and amazon.com in 2000. ETH itself was the target of such an attack six months before these commercial sites where hit. ETH suffered repeated complete loss of Internet connectivity ranging from minutes to hours in duration. Massive DDoS attacks have the potential to cause major disruption of Internet functionality up to severely decreasing backbone availability.

The DDoSVax Project In the joint ETH/SWITCH research project “DDoSVax” [3] flow-level Internet traffic data (Cisco Netflow V5) is collected at all four border gateway routers operated by SWITCH. This data contains information about which Internet hosts were connected to which others and how much data was exchanged over which protocols.

Netflow data evaluation All the gathered data by the DDoSVax Project is stored on large disks post processing. There is a wide range of possible evaluations from near real-time statistics over the past 15 minutes to larger analyses over several months. The later of these two scenarios is a job for a cluster like Scylla [5]. We focus in our work on near real-time plots for a network operator, so he can get an impression what is going on in his network. These plots should help to recognise DDoS-like attacks.

1.2 Task description

A generic robust real-time UDP Netflow (Internet connection data) processing framework called UPFrame [1] was developed in a diploma thesis. With this framework it is possible to collect the exported Netflow data from several border gateway routers and forward the records with smoothed traffic bursts to data processing plug-ins. All plug-ins running under the framework can be monitored by a watchdog process to restart hanging or crashed plug-ins.

The task is split into three major subtasks: understand the Netflow data and its possibilities, implement a sample plug-in like flow size distribution, and develop one or two own attack detection methods.

Understand the Netflow data and its possibilities In a first step it is required to understand how the relevant data is stored in a Netflow record and how it can be used to detect attacks.

Implement a sample plug-in like flow size distribution To get familiar with UPFrame and its plug-in support it would be wise to start by writing this simple data gathering plug-in. It is also a good starting point to see which performance bottlenecks in real-time processing must be considered.

Develop one or two own attack detection methods A common pattern of DDoS attack is the increased traffic originating at hosts which where mostly inactive in the past. Analysing Netflow records and collecting traffic statistics per IP subnet should expose these attacking hosts. In combination with filtering Netflow records by port numbers (e.g: 25 for Internet worms implementing their own SMTP MTA) this gives a powerful tool to recognise DDoS-like attacks.

1.3 Related work

There are a number of Netflow data processing and analysing program suites available.

One of them is SiLK [6], a System for Internet Level Knowledge, developed at Berkeley's CERT and released to the public in early 2004. It helps in packing and analysing Netflow data. The development focus was on providing a space efficient format for storing Netflow data and the tools to query them. For the query part, the basic idea is, to read in the Netflow data which was compressed by the data storing part and pipe it through different analysis tools. `rwfilter` is to filter the Netflow data by criteria applying to the fields of a Netflow record. `rwcount` can generate aggregated traffic statistics on fields of a Netflow record. `rwstat` looks for the most/less active hosts, ports or IP protocols. There are some other tools for sorting records or working with sets of extracted IP addresses. This tools are handy when you detected some traffic anomaly and want to find the exact cause of it.

Another tool chain is cflowd [7] from CAIDA [4] and its successor flow-tools [8]. This one seems better for DDoS detection, the program `flow-dscan` can detect portscans and `flow-host-profile` builds host profiles to detected new services like worms with their own SMTP engine. However, cflowd is no longer supported.

Chapter 2

Specification

We want to support a network operator in detecting attacks. An obvious characteristic of a DDoS attack is a repeating pattern of the same network packets, often even with the same size. Our first plug-in tries to visualise this by creating a histogram plot over all possible flow sizes[1].

A lot of attacks use some kind of scanning the detect vulnerable hosts. So large fields of the inactive IP address space are now receiving packets from attacking hosts (either with a real or a spoofed source address). The IP activity plot should be some kind of map of the IP address space and give an impression of the activity of each host.

2.1 Flow-Size distribution plot

For a given time period all flows of the same size are counted and plotted in a histogram plot. The variation over time can be visualised by making an animation out of single 60 seconds frames or drawing a 3D plot with time as an additional axis. Ongoing attacks using a fixed packet size (e.g: SQL Slammer, 404 bytes) are now visible as large peaks in the histogram plot. The user interface is both a command line program for batch plot generation and a web interface for interactive use.

2.2 IP-Activity plot

Representing each 32 bit IPv4 address as one pixel would result in an image with dimensions larger than 65000 times 65000. This leads to grouping them in "virtual" sub-networks. "Virtual" in the sense that we do not follow the definition of subnets in [2], instead we just group IP addresses with the same n most significant bits together. With this technique we can draw the whole IP space in a plot with 1024 times 512 pixels by erasing all the 13 least significant bits of every IP address.

The color value of each pixel represents the amount of data sent or received by this host group in the given time period. This allows easy differentiation of active and passive hosts and can expose scanning traffic by highlighting clusters of adjacent pixels.

The user interface is implemented in the same way like in the Flow-Size plot. An additional feature is a clickable map to allow zooming and navigation into the IP address space.

[1] In Netflow terms, a flow is unidirectional and groups all subsequent IP packets identified by seven unique keys (source IP address, destination IP address, source port, destination port, layer 3 protocol type, type-of-service byte, incoming router interface).

Chapter 3

Design

There are two different design approaches to discuss, either a monolithic UPFrame plug-in for data processing and data visualisation or split these two tasks into different programs.

Monolithic plug-in One UPFrame plug-in is responsible for receiving the Netflow stream, doing the necessary analyses and providing a graphical visualisation for the user. This would require to run the plug-in on the UPFrame host and export the plots to some network storage. All data for plotting needs to be hold in a memory buffer and rotated from time to time, when new data from UPFrame comes in. This implies that everything is calculated in real-time and just a plot over the last few minutes can be presented to an operator. Interaction with the plug-in is hard to achieve, reconfiguration with some other plotting parameters needs a restart of the plug-in.

Separated processing and visualisation Another approach is to make the UPFrame plug-in as small and fast as possible. Netflow data received is analysed and just the relevant parts are written to a network disk for later visualisation. A user driven visualisation program just calculates the needed images on demand on another host. So the UPFrame host is not affected by larger plotting tasks. It is not only possible to make plots from the last few minutes but also from periods further back in time. Different plotting parameters can be tried out without interrupting the data processing plug-in.

There are several advantages of a monolithic plug-in, data flow and storage can be optimised, most parts of the plug-in get simpler and less error prone. On the other hand it is a drawback that just a plot of the last few minutes is provided to the operator and there is no possibility to choose other time periods because everything is done in real-time.

This separation approach has the downside that large statistic files need to be kept on disk for later visualisation and that the CPU and I/O usage is not to be neglected. The benefits of the design are the clear separation between data processing and visualisation, the possibility to plot arbitrary time periods and an easy changement of plotting parameters.

The second design proposal is taken because of its better user interaction and modularity.

3.1 Data processing part

As you can see in Fig. 3.1 the Netflow streams exported at all four SWITCH border gateway routers[1] are collected by a running instance of UPFrame. Every interested process can now register in UPFrame to receive a copy of the Netflow data stream. In our case this is the Flow-Size and IP-Activity plug-in. Another possibility is to run these two plug-ins in standalone mode and read the Netflow data in from archived files as shown in Fig. 3.2. This is convenient for offline processing or debugging purposes.

[1] At the time of writing these are: swiIX1, swiBA2, swiCE2 and swiCE3

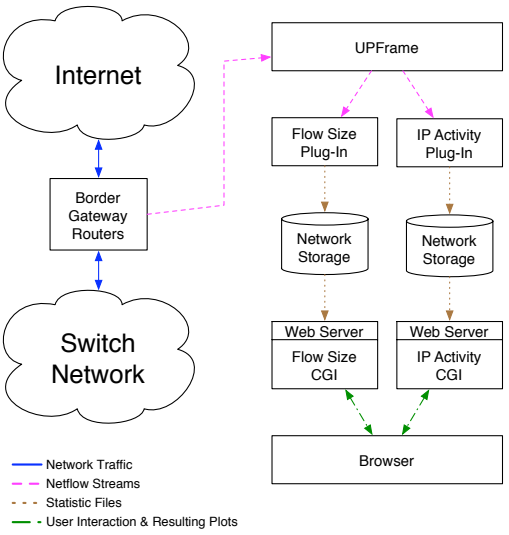

Figure 3.1: Plug-in design

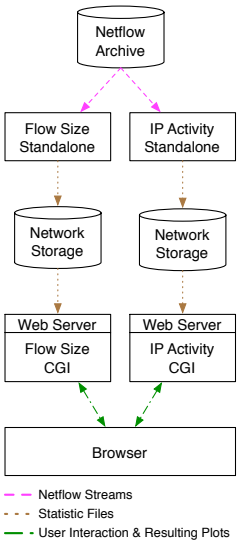

Figure 3.2: Standalone design

In a next step, each plug-in splits the Netflow stream into different traffic classes like TCP-In/Out, UDP-In/Out etc. During a configurable time period (e.g.: 60 seconds) the analysed data is kept in memory and aggregated.

3.1.1 Flow-Size

To create a flow size histogram, the corresponding plug-in counts during the aggregation period the Netflow flows with the same length. Still separated in the different traffic classes. After the end of an aggregation period, the counted flows per flow size are written to disk.

3.1.2 IP-Activity

The traffic exchanged by each unique IP address during the aggregation period is added together. A total of transferred traffic per IP address is written to disk after each aggregation period.

The aggregated data is stored on disk for later visualisation. This is done on a per traffic class basis and compression can be applied to reduce file size.

3.2 Data visualisation part

The data visualisation part is user driven. A user can specify an aggregation period and additional plotting parameters. The corresponding statistic files are read from disk and if needed decompressed on the fly. If the user chosen aggregation period is larger than the period chosen in the plug-in, data needs to be re-aggregated to the new time period. This and other plotting parameters can be entered in a web interface, the resulting plot is then returned by the CGI program.

3.2.1 Flow-Size

The Flow-Size plot is a simple histgram plot and a pre-existing software package like `gri` [11], `gnuplot` [12] or `plplot` [13] can be taken for this task. The user has the option to apply logarithmic scaling either to the x or y axis. To visualise changes over time in the Flow-Size distribution, multiple plots over 60 seconds can be joined together and displayed as an animation.

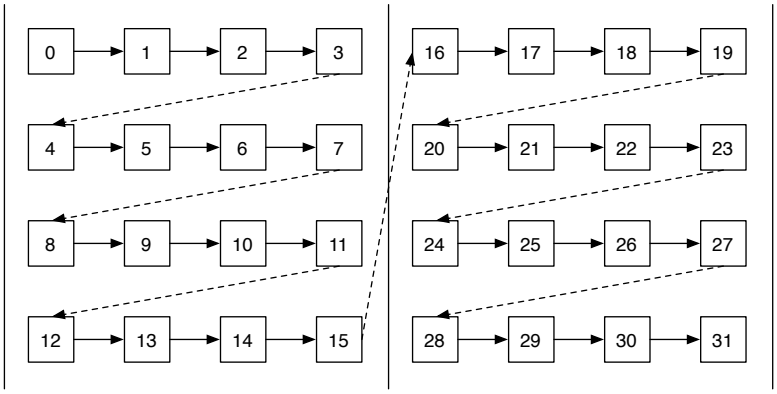

Figure 3.3: IP-Activity plot organised in stripes

3.2.2 IP-Activity

The IP-Activity plot is a XY-plot. Aligning all virtual subnetworks[2] from left to right and from top to bottom can place the start and end pixel of a real subnetwork far away from each other. Zooming into a rectangular area of the plot would not make that much sense. Every horizontal pixel line can belong to a different subnetwork or just a part of a subnetwork. These pixel lines are now isolated and independent of each other.

It would be better to keep adjacent virtual subnetworks closer together. Closer in the sense to minimise the diameter of the bounding box of all pixels of one real subnetwork. As seen in Fig. 3.3 we organised our plot in stripes to bring neighbouring subnetworks closer together.

To be flexible enough to make stripes at our need, we implemented the plotting part and did not rely on an existing plotting framework. The plot is created on the fly from the filtered Netflow data. Filtering is done to concentrate on a particular IP range or time period.

The most active virtual subnetworks are highlighted by a surrounding circle with diameter proportional to the traffic they sent.

[2] One pixel corresponds to one virtual subnetwork

Chapter 4

Implementation

UPFrame is written in C, so it is obvious to write our plug-in also in C. We chose C++ for the plug-in part. The STL provides some of the abstract data types we need, like hash tables, linear lists etc. Some of the object oriented features were used to simplify code reuse. The CGI part responsible for data visualisation, more or less needs to read data from disk, redirect it to a plotting framework and interact with the user as a CGI program. Perl and PHP provide both these features. We chose Perl because it is shipped with most UNIX like operation systems, and is better suited to also write a command line program.

4.1 Data processing part

4.1.1 Data buffering

Netflow data is coming-in in a more or less ordered way. But it can happen, just before we advanced from one aggregation period to the next one, that we receive interleaved data of these two periods. To bring back a chronological order, we store our aggregation periods in a ring buffer as shown in Fig. 4.1. For most cases, it is enough to have the three most recent periods in memory. A new starting period queues out the oldest one and triggers to write it to disk. Every flow is counted with its end time. Neither the Flow-Size nor the IP-Activity statistic averages the traffic of one flow over its duration. Flows would lose their characteristics and attacks using the same packet size but the sending of them is dragged on over several minutes could not be recognised anymore. IP-Activity plots should be drawn over less than 15 minutes[1] and averaging is so not necessary.

4.1.2 Flow and traffic counting

We need to sum up the traffic of every unique IP address, during an aggregation period. Using an array with fields for each of the 2^{32} IP addresses would require something of about 32 GBytes memory[2]. Instead a hash table is better suited for this task. So we only need to store the really seen IP addresses. To avoid the expensive allocated memory resize operation, we initially grow the table size to some average we have seen during several program runs. This maybe needs to be changed when running with different traffic loads.

Traffic is counted at the source and at the destination side. This double traffic counting allows us to see in the traffic class TCP/UDP/Other-In all hosts sending traffic to SWITCH networks and all hosts inside SWITCH networks receiving this traffic (and vice versa with TCP/UDP/Other-Out).

[1] Flows larger than 15 minutes are reported by the router after this period. A new flow is opened and sampling continues. This 15 minute period is configurable at the router.

[2] We have 2^{32} IP Addresses, each of them needs 4 Bytes and we use a 4 Byte integer for the summed up traffic of each IP. $2^{32} * (4 + 4) = 32$ GBytes.

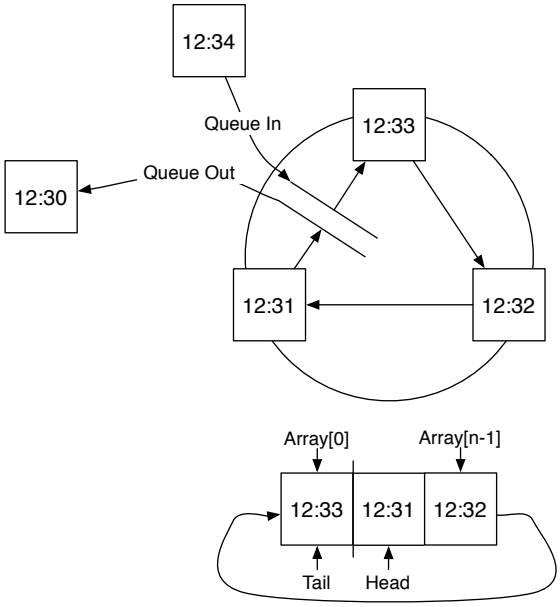

Figure 4.1: Ring buffer with 60 seconds aggregation period and the corresponding flat array

To make an example for better understanding, we send 128 byte TCP traffic from host A to host B. A is outside of the SWITCH network, B is inside. This results in an incoming flow (TCP-In). As we count traffic at the sending and receiving side, we increment the traffic counters for A and B by 128. In plots showing both A and B (like in Fig. 6.6) we have two pixels for this flow, one at the sending side A and one at the receiving side B.

This double traffic counting allows us to see in the traffic class TCP/UDP/Other-In all hosts sending traffic to SWITCH networks and all hosts inside SWITCH networks receiving this traffic (and vice versa with TCP/UDP/Other-Out).

For flow counting it is mostly the same. The flow size is used as the key in the hash table and the value is how many times this flow size has occurred.

4.1.3 Statistic files

After an aggregation period is queued out by the ring buffer, it is written to disk. For later visualisation, we need the following information stored in these files: start and end time of the aggregation, aggregation duration and the data part. For the Flow-Size plug-in, we have pairs of the flow size in bytes and how many times it occurred. Flow sizes that did not occur, are left out in the statistic file. For the IP-Activity plug-in we have pairs of IP addresses and the traffic in bytes they transferred. We used the following directory structure:
`./<basedir>/<year><month>/<day><hour>.stat`, every file contains the corresponding aggregations of this hour. Before we start writing to a file, we check it's existence and add if needed an incremented number at the end. Several of these files can be merged with `netflowMerge`.

The file format could either be binary or ASCII. Binary files have the advantage that they are in our case about 3 times smaller than the ASCII counterpart and parsing is less time consuming. An ASCII representation is human readable and easier to debug, its file size deficiency can be compensated with compression. Because it is also simpler to read in ASCII instead of binary files in Perl we chose ASCII as our statistic file format. The exact file format is described in Sect. A.3.

All files are compressed with `lzop` [16], and the suffix becomes `.stat.lzo`.

4.1.4 Traffic classes

We differentiate 6 traffic classes, TCP-In/Out, UDP-In/Out and Other-In/Out. In and Out is enough because we are collecting Netflow data at the border gateway routers and intra traffic does not go through this routers. We see just a small fraction of intra and transfer traffic which addresses the router itself (like SNMP) or passes through the AS and hence there is no need for extra traffic classes. Traffic class filtering is done by a list of the network addresses of all subnets SWITCH is responsible for and are therefore part of AS559.

4.2 Data visualisation part

4.2.1 Flow-Size

As plotting framework we chose `gnuplot`. Data is read in from disk by our Perl program, decompressed, re-aggregated and piped to `gnuplot`. A bidirectional pipe is used to run `gnuplot`, so the resulting image written to stdout by `gnuplot` is redirected to ImageMagick [15] for post-processing. To create the animation over time, we loop with an increment of 60 seconds over the plotting period and join all images together with ImageMagick to an animated GIF. It is important to use an ImageMagick version with activated LZW support[3], animation files would otherwise be larger by a factor of 10.

4.2.2 IP-Activity

The IP-Activity visualisation program is written in Perl like the Flow-Size visualisation and parsing these long lines of IP addresses with regular expressions (like the `split` operator in Perl) is very time consuming. To speed up the reading of the IP-Activity statistic files, a pre-converter in C was written to place every IP address on a single line. The Perl program pipes the decompressed text files through this converter and reads it in after the conversion.

To draw the custom IP-Activity plots, the GD Graphics Library [14] is used. The color values of each pixel represents the traffic transferred by this virtual IP address. All traffic values are first log^2 scaled to values between 0 and 1. The color value is then looked up in a colormap with 64 entries. Inactive virtual IP's are black, low activity is from red to yellow and middle to high activity goes from light blue to blue.

4.3 Performance

To give a rough idea of the memory and CPU consumption of the Flow-Size and IP-Activity plug-in, we replayed an archived Netflow data hour during the time the Nachi worm was active. The compressed file is about 870 MBytes large and expands during uncompressing to 2.7 GBytes. It contains about 75 million Netflow records, 465 GBytes of transferred traffic are covered and 6 million unique IP addresses occurred. The Netflow data was captured in December 2003 on a weekday in noontime.

All performance tests were done on a Athlon 1.4 GHz system, 256 MBytes memory and running GNU/Linux with Kernel 2.4.26. Running both programs in standalone mode gives the following execution times:

Program	Wall Time (s)	User Time (s)	System Time (s)
Flow-Size	108	58	15
IP-Activity	239	185	28

[3] Using Debian GNU/Linux, you need to recompile the ImageMagick package and activate LZW support in `<ImageMagickSource>/debian/rules`.

To see how these plug-ins integrate with UPFrame and to estimate how many other plug-ins can be run on the same host, the Netflow data was replayed in real-time and processed by UPFrame. The CPU and memory usage was continuously sampled from `/proc/*/stat` every half a second.

In Fig. 4.2 and 4.4 the CPU usage in percent is plotted along the running time of the plug-ins. To smooth the sampled CPU data, a Savitsky-Golay filter (Listing 4.1) was applied 10 times with a filter width of 20. The CPU execution time is linear depending on the number of Netflow records to process.

```
% savitsky-golay filter as shown in the scientific computing lecture held by
% prof. gander in the summer semester 2002 at ethz
%
% y: data vector to filter
% d: filter width
% n: apply filter n times
%
% yf: filtered vector
function yf = savgol(y, d, n)

M = length(y);

a1 = round(d / 2); a2 = d - a1;

yf = [zeros(1, a1) y zeros(1, a2)];

for i = 1:n

    ly = yf;
    Mly = length(ly);

    for j = 1:(Mly - d)
        lyf(j) = 0;

        for k = j:(j + (d - 1))
            lyf(j) = lyf(j) + (1 / d) * ly(k);
        end
    end

    yf = lyf;
end

yf(1) = y(1); yf(M) = y(M);
```

Listing 4.1: savgol.m

The memory usage shown in Fig. 4.3 and 4.5 is more or less constant during execution. The Flow-Size and IP-Activity programs need about 8 MBytes and 48 MBytes respectively. Memory usage is a linear function of the number of unique flow sizes and unique IP addresses for the Flow-Size and IP-Activity plug-in respectively.

Running the plug-ins under heavy scanning traffic load with lots of unique IP addresses, can increase the memory requirement dramatically. To prevent crashing the whole system, upper memory consumption limits need to be implemented. This is open to further development.

The storage requirement for the compressed statistic files lies between 12 MBytes for the Flow-Size plug-in and 240 MBytes for the IP-Activity plug-in. This is just for one hour. You need to take care when running the plug-ins unattended for several hours and delete statistic files not needed anymore.

There was no closer consideration of the CGI visualisation programs. CPU and memory usage are heavy depending on the time period and IP range to plot. And in addition, these two programs are run on another host than UPFrame and do not interfere the real-time capability of the plug-ins.

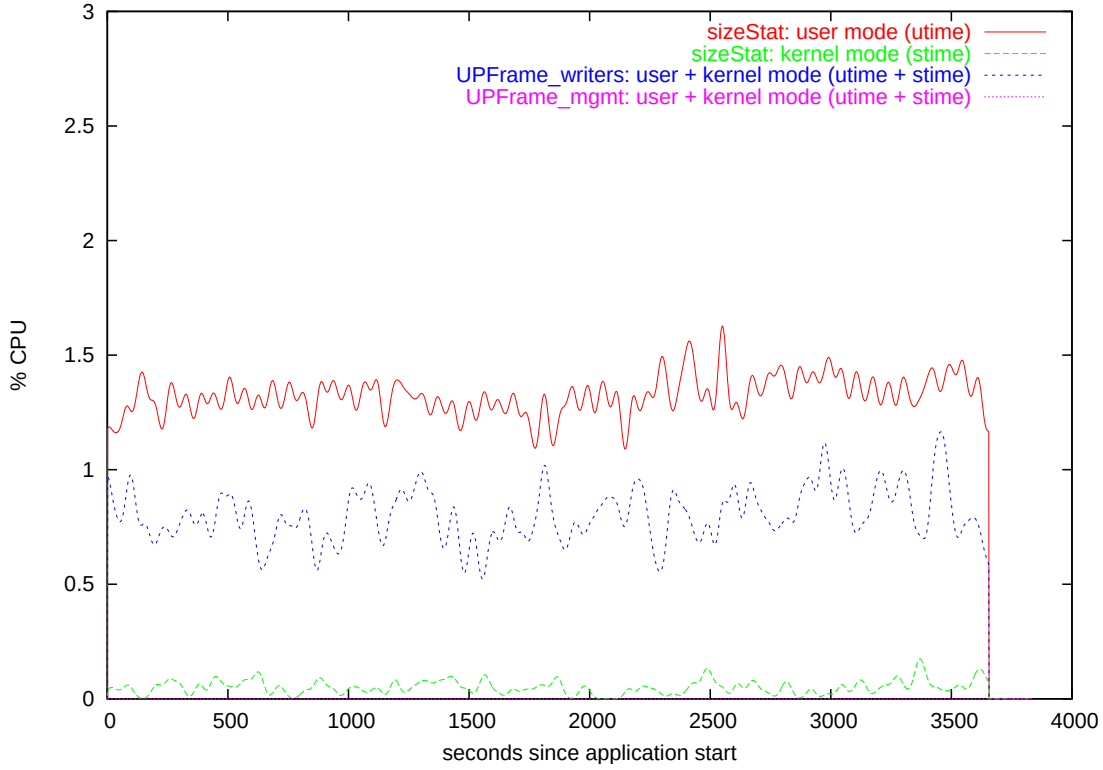

Figure 4.2: CPU usage of the Flow-Size plug-in

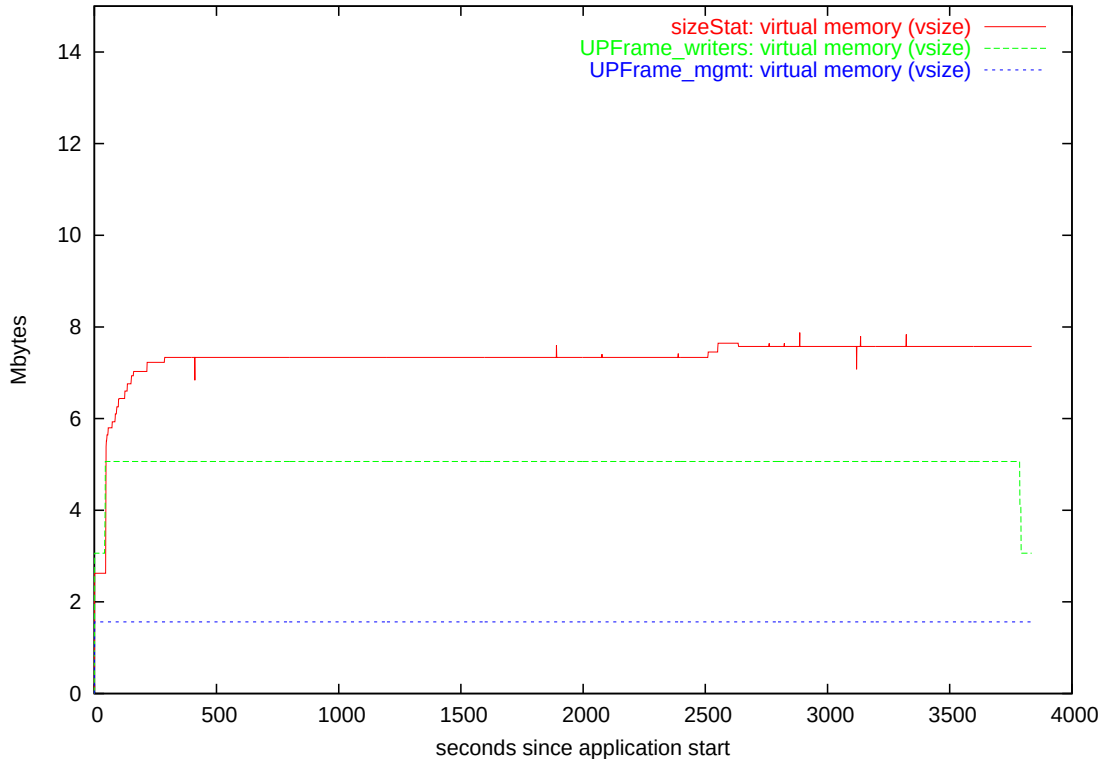

Figure 4.3: Memory usage of the Flow-Size plug-in

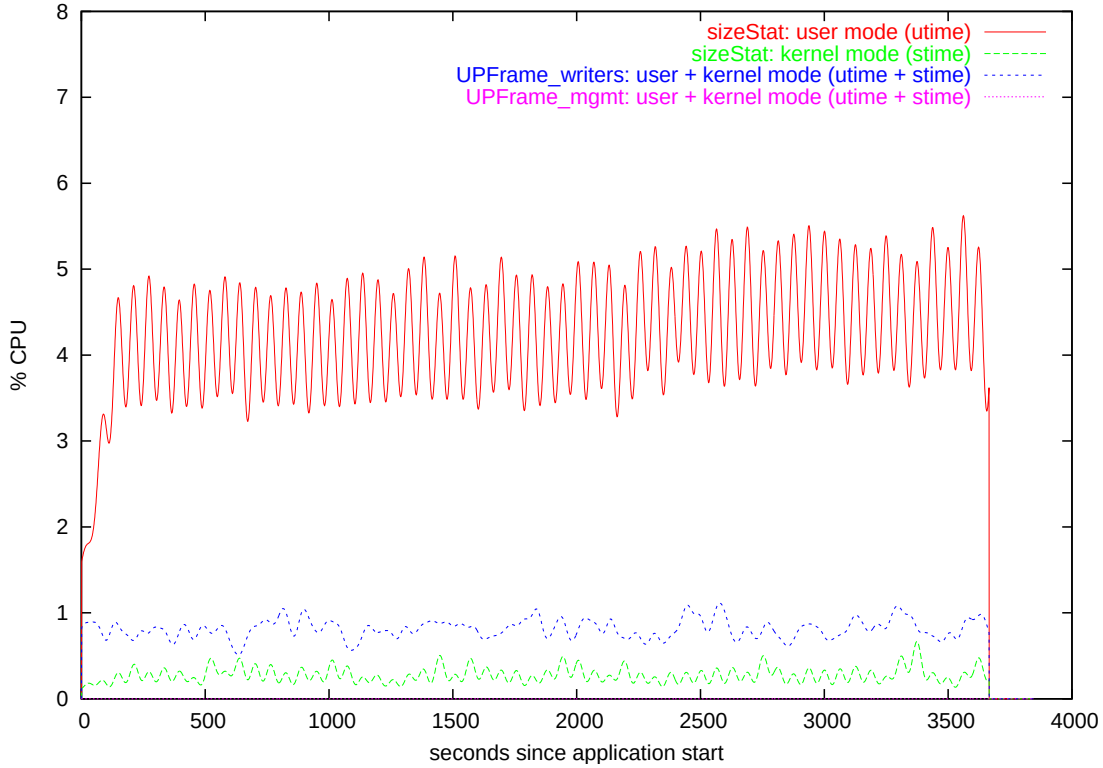

Figure 4.4: CPU usage of the IP-Activity plug-in

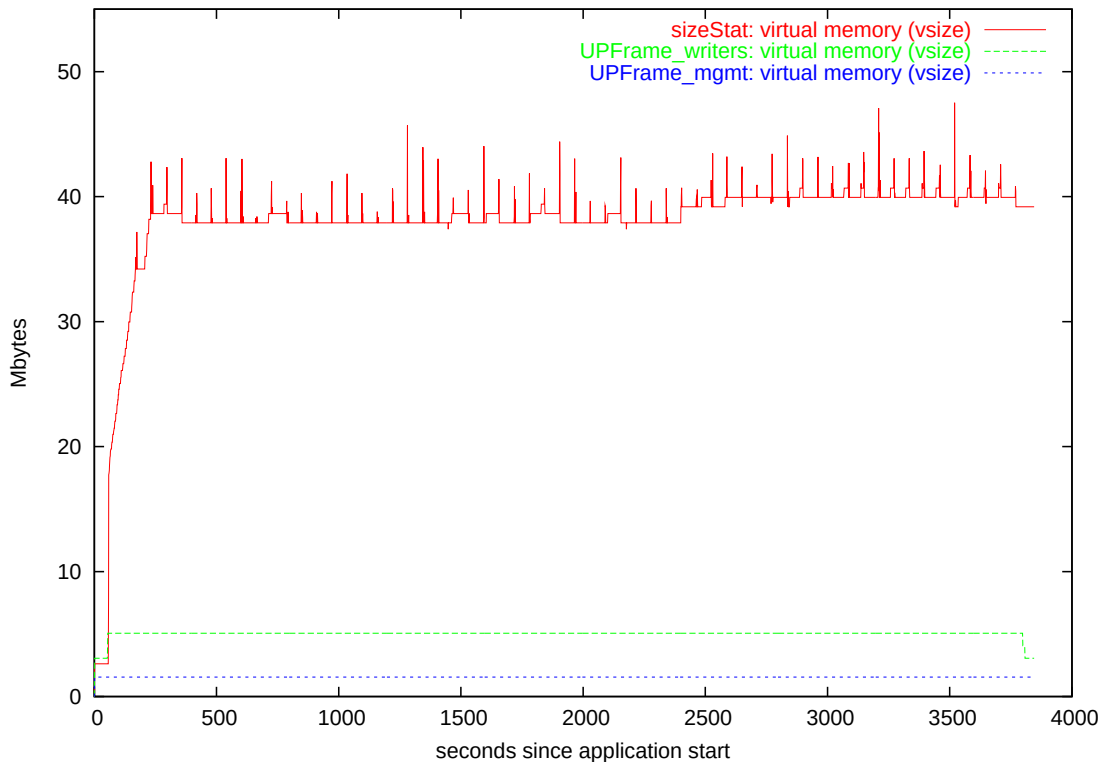

Figure 4.5: Memory usage of the IP-Activity plug-in

Chapter 5

Testing and Validation

Basic testing and validation of the data processing and visualisation part were done by injecting custom traffic patterns and try to recognise them in the plots. Traffic was generated using a Perl script and `packit` [10]. The resulting Netflow records were generated with `nProbe`[1] [9]. The traffic was not sent out to the Internet, it was generated locally on one host and captured afterwards. Firewall rules prevented them to be sent out to the Internet.

5.1 Custom Flow-Size peaks

We sent to every ETH core network IP[2] one 64 byte, two 128 byte and three 256 byte TCP packets. This should result and results in three spikes in the Flow-Size plot as seen in Fig. 5.1.

5.2 Scanning traffic

Starting at 129.132.0.50, we sent to every 255th IP address[3] one packet. This results in diagonal lines of activity in the IP-Activity plot in Fig. 5.2.

5.3 IP spoofing traffic

Common in DDoS attacks is the spoofing of the sender IP address. To simulate this traffic pattern, we sent packets with randomly chosen source address[4] to the ETH network. Figure 5.3 shows normal activity. This was overlaid with the spoofed packets and is shown in Fig. 5.4. Compared to the original plot, this one is covered with dense noise caused by the randomly chosen source addresses.

[1] We used version 3.0.3 and had severe problems. After several hours trying different settings, we still had a flow loss rate of 25% when exporting to port 2055 over the loopback interface `lo`.

[2] 129.132.x.x: 2^{16} IP addresses

[3] $startIP = dot2dec(\text{"}129.132.0.50\text{"}) = 2172911666, IP_i = startIP + (255 * i), i \in [0, 255]$

[4] done with `packit -sR`

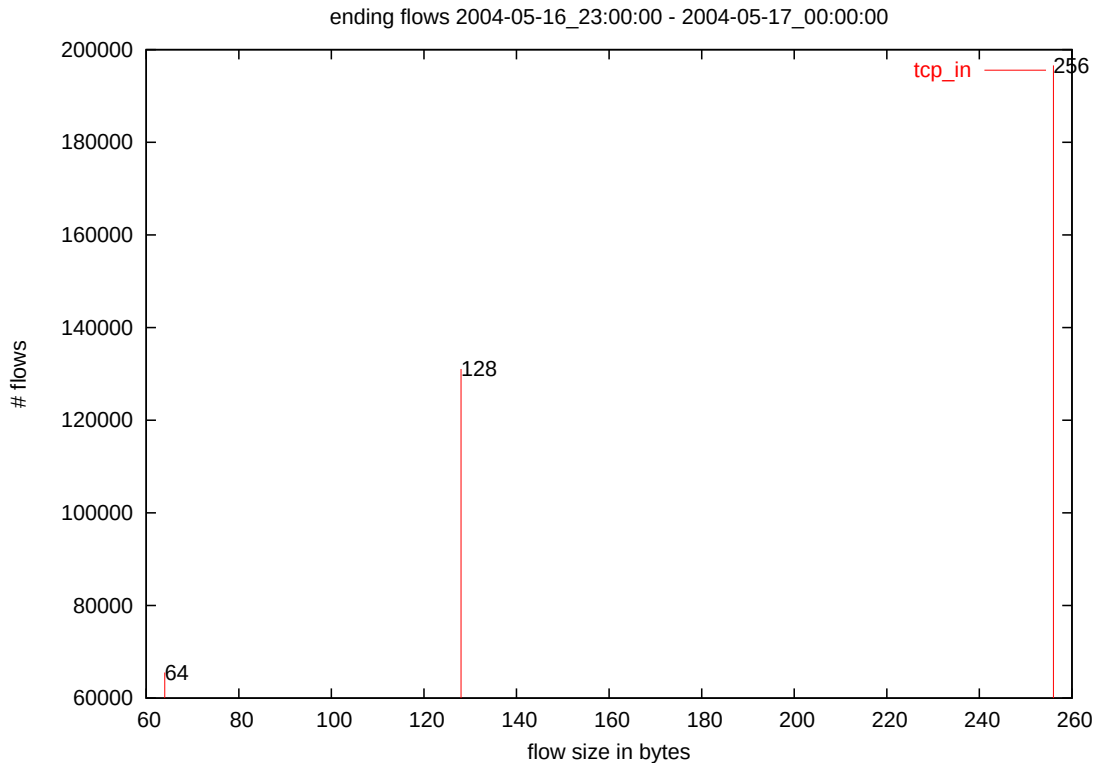

Figure 5.1: Custom flow size peaks at 64, 128 and 256 Bytes flow size

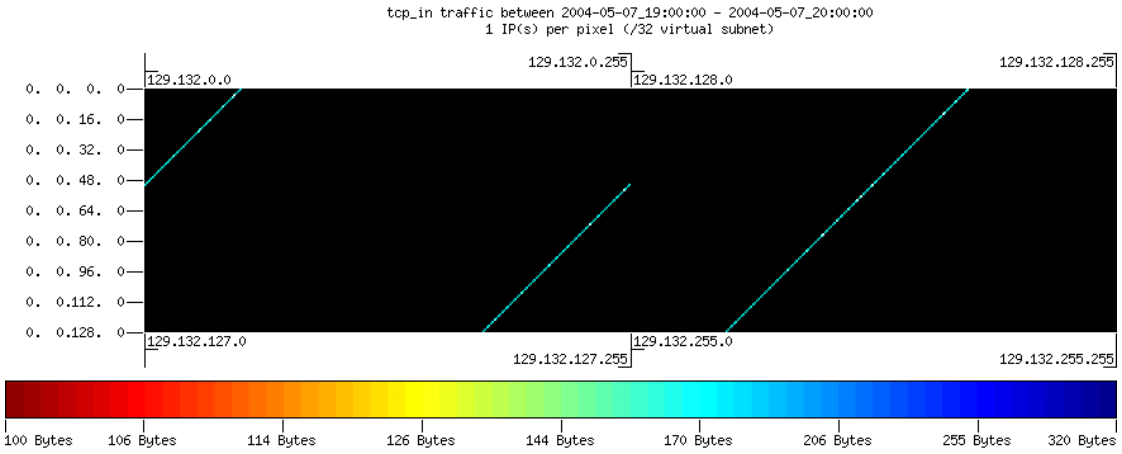

Figure 5.2: Scanning traffic resulting in diagonal activity lines

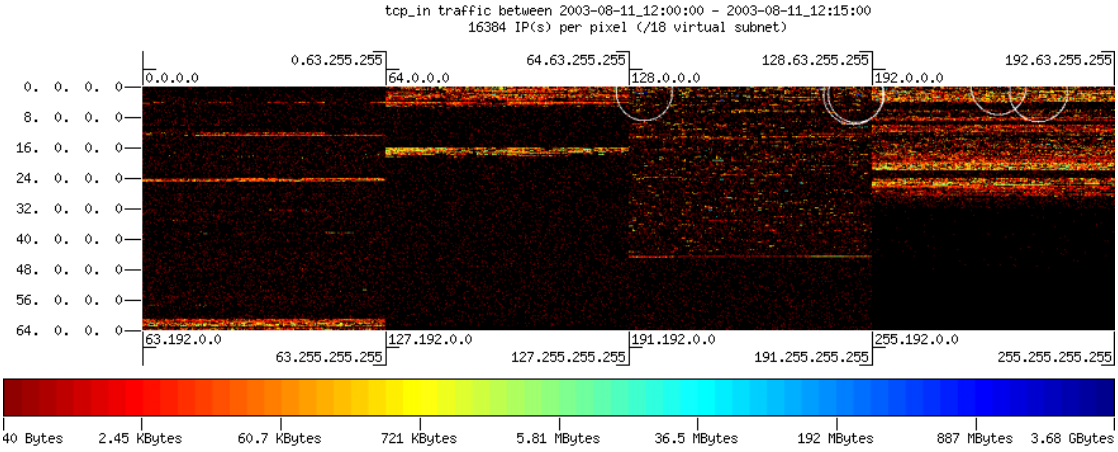

Figure 5.3: Normal IP-Activity plot

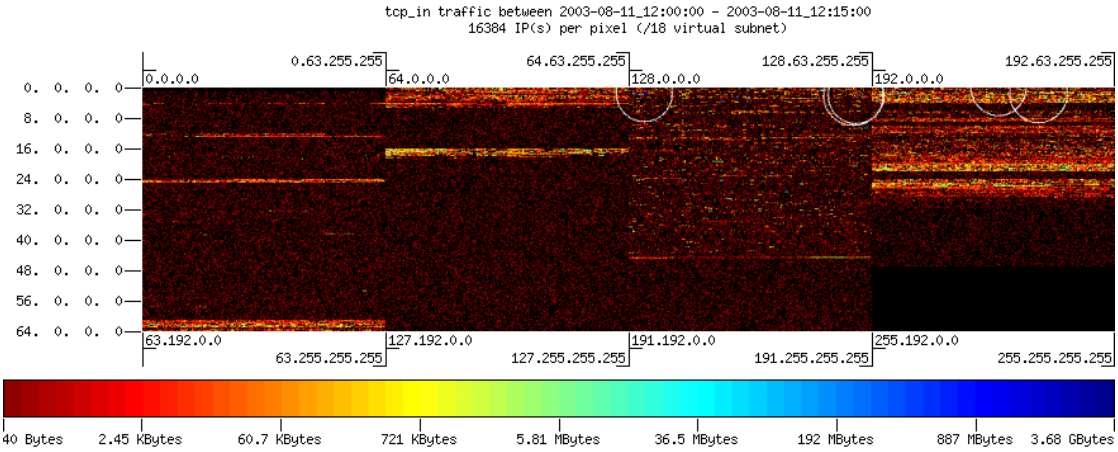

Figure 5.4: IP-Activity plot exposing IP source address spoofing

Chapter 6

Conclusion

6.1 Results

During this semester thesis, two UPFrame plug-ins evolved. They try to help a network operator to see what is going on in his network. The postulated real-time capability is achieved, the Flow-Size plug-in is running on a Athlon 1.4 GHz system about 30 times faster than real-time, the IP activity plug-in performs 15 times faster.

6.1.1 Flow-Size

The Flow-Size visualisation plug-in gives a compact description of the actual flow size distribution in the monitored network. Some typical attack scenarios are identifiable. In Fig. 6.1 you can see the flow size distribution during noontime of a normal working day. Some hours after W32-Blaster started his activity, the picture changed dramatically. W32-Blaster tries to infect hosts with a vulnerable Windows RPC implementation and sends so many TCP connection attempts to port 135, identifiable as the 48 byte peak in Fig. 6.2 which is 10 times larger than before.

The same picture with Nachi in December 2003. In Fig. 6.3 we see normal traffic in the class Other-In (everything except TCP and UDP) during noontime on a week-day. After Nachi started his activity in Fig. 6.4, we have a peak at 92 bytes which was non existent before.

6.1.2 IP-Activity

With the IP-Activity plug-in, a network operator has a quick overview over the traffic going through his border gateway routers.

Otherwise as stated in the task description, the visualisation over time was not achieved. We assumed a binomial traffic distribution but found a heavy tailed distribution as shown in Fig. 6.5. This makes it impossible to use the standard deviation to derive changements over time. We delayed the search for a better mass to a maybe written follow-up thesis.

We see in Fig. 6.6 an overview of all IP addresses communicating (inbound traffic) with the SWITCH network. This kind of map can expose very active network segments and typical DDoS scanning traffic.

The increased traffic produced by W32-Blaster can be seen in Fig. 6.8 in comparison to the normal work load in Fig. 6.7.

Nachi scans its neighboring networks and sends out custom made ICMP packets to those hosts. This scanning traffic is visible in Fig. 6.10. The two vertical and one horizontal black lines in the map are due to some internals of the worm which makes it not possible to Nachi to scan IP addresses containing the byte `0xC3` (decimal: 179).

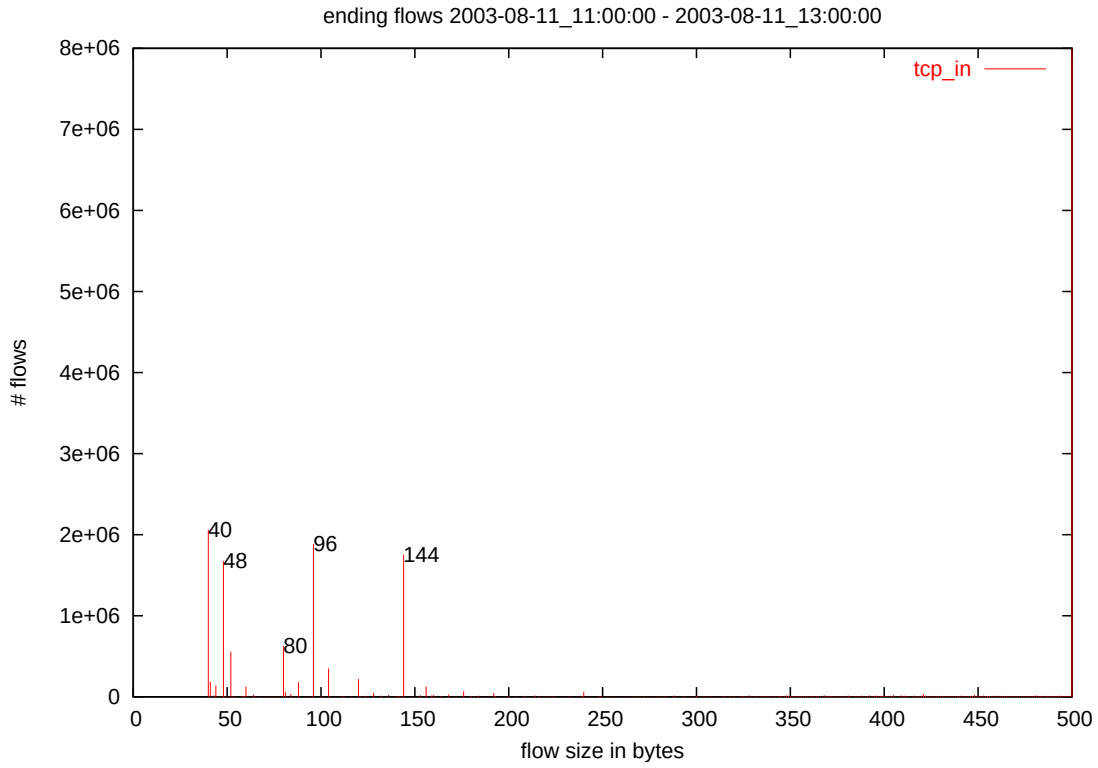

Figure 6.1: Flow size distribution (TCP-In) on a normal work-day

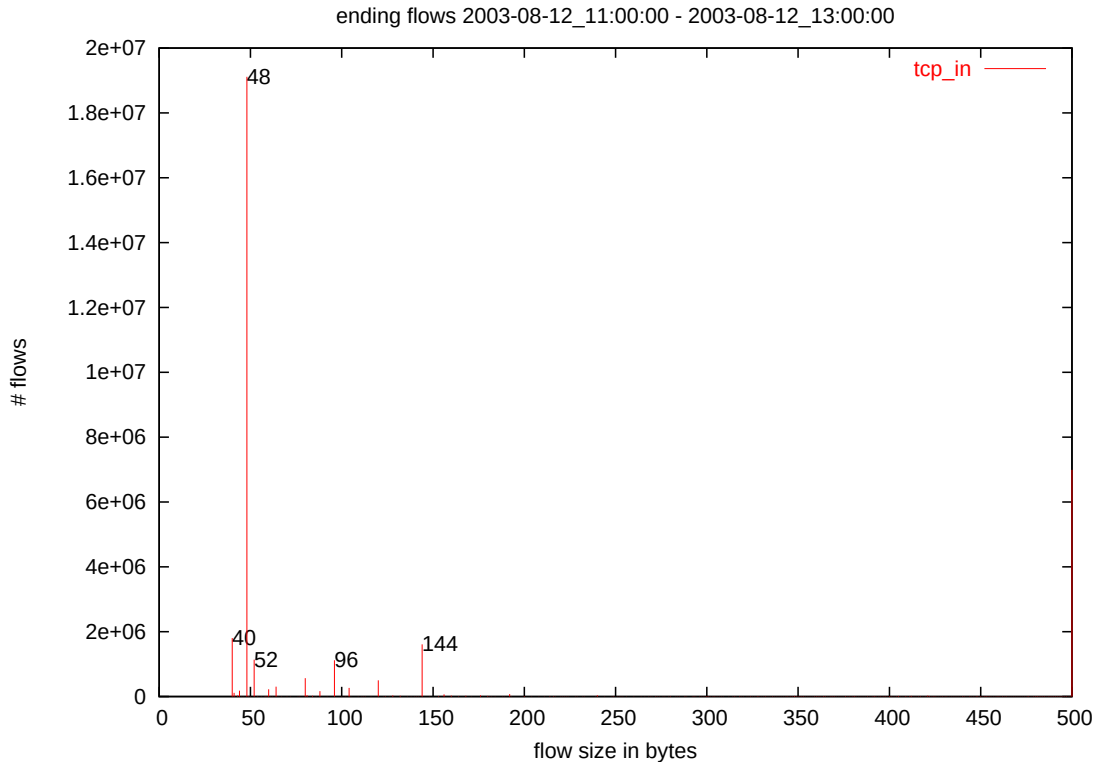

Figure 6.2: Flow size distribution (TCP-In) on a work-day for W32-Blaster

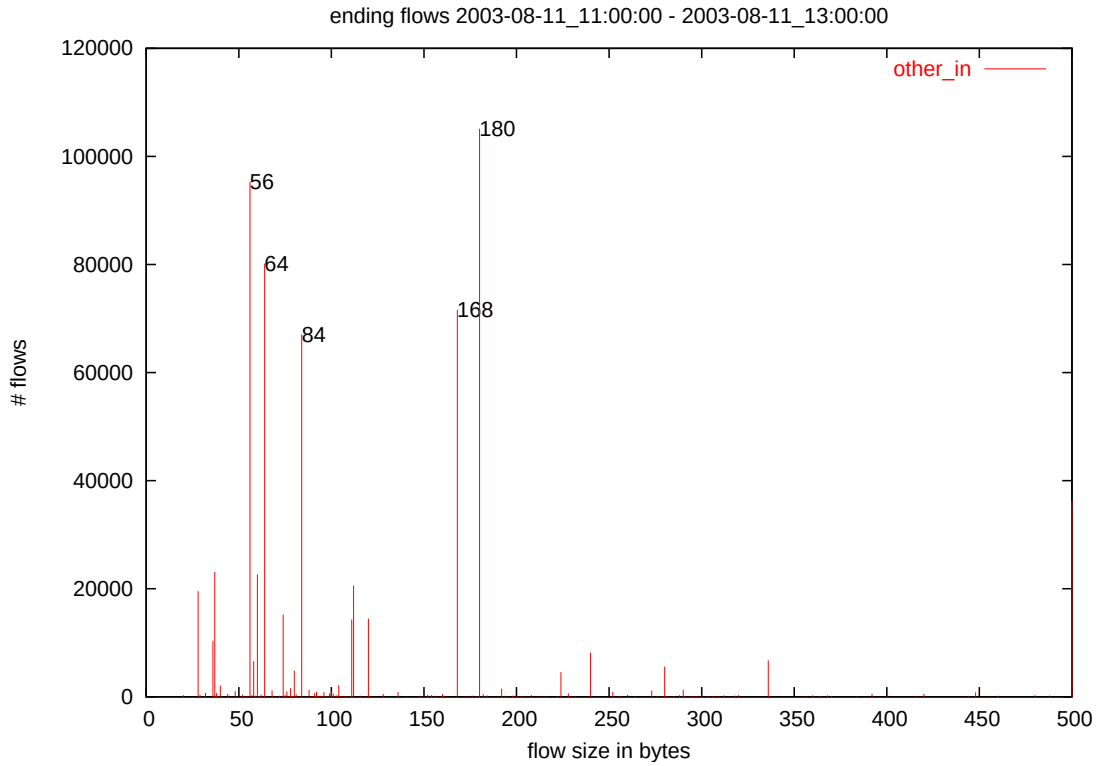

Figure 6.3: Flow size distribution (Other-In) on a normal work-day

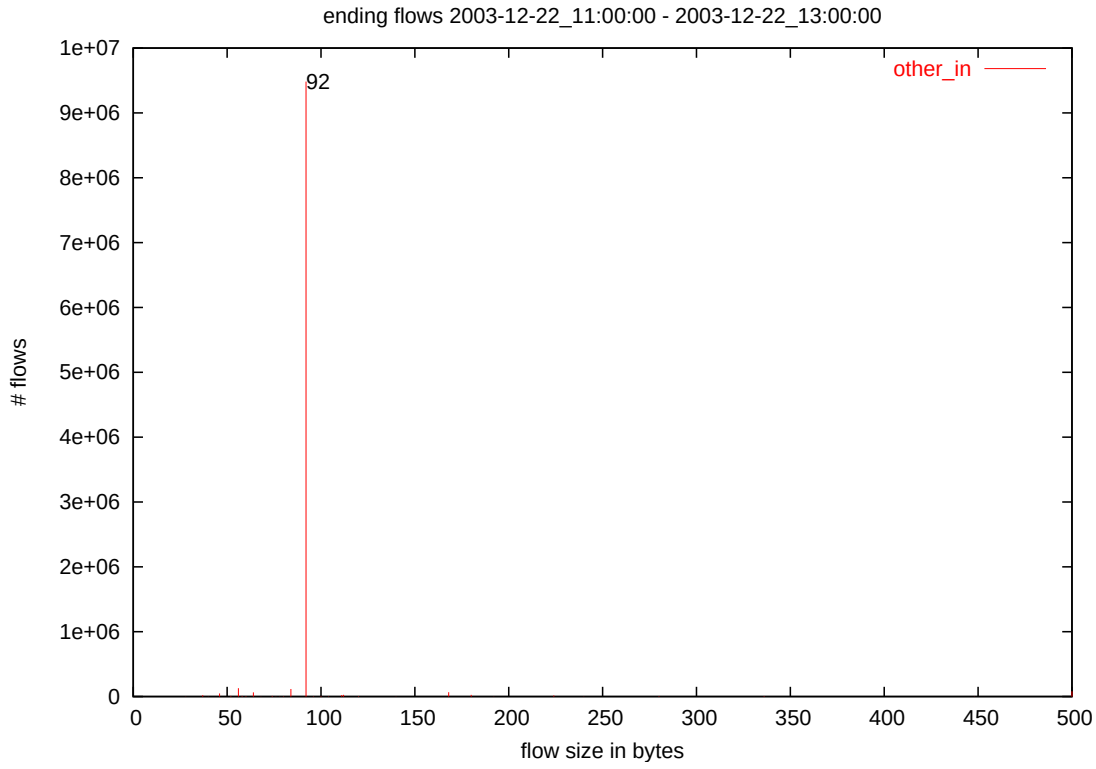

Figure 6.4: Flow size distribution (Other-In) on a work-day for Nachi

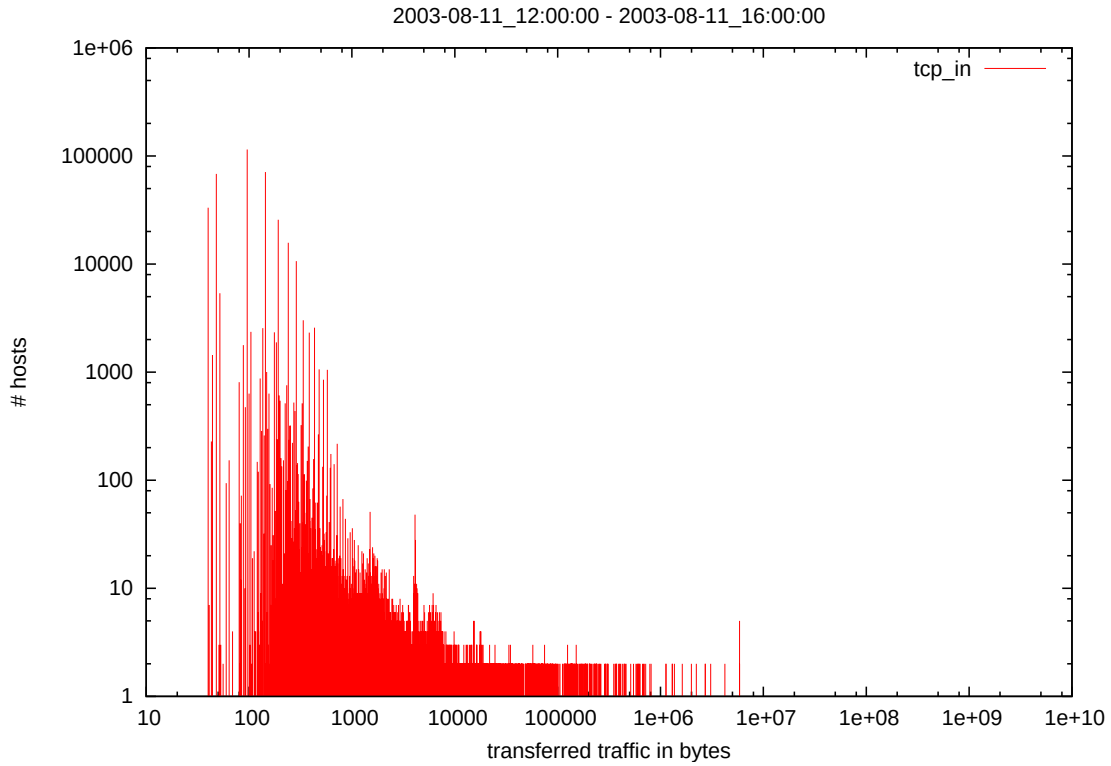

Figure 6.5: Heavy tailed transfer statistic

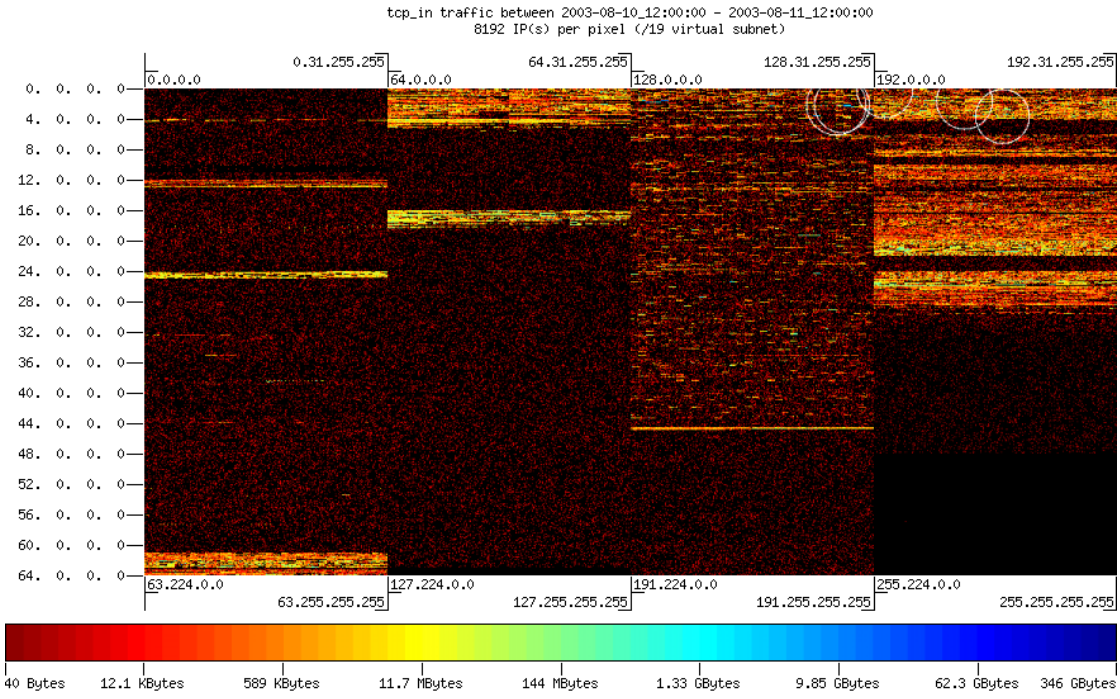

Figure 6.6: Inbound traffic to the SWITCH network for one day. The 5 most active hosts are highlighted with a white circle.

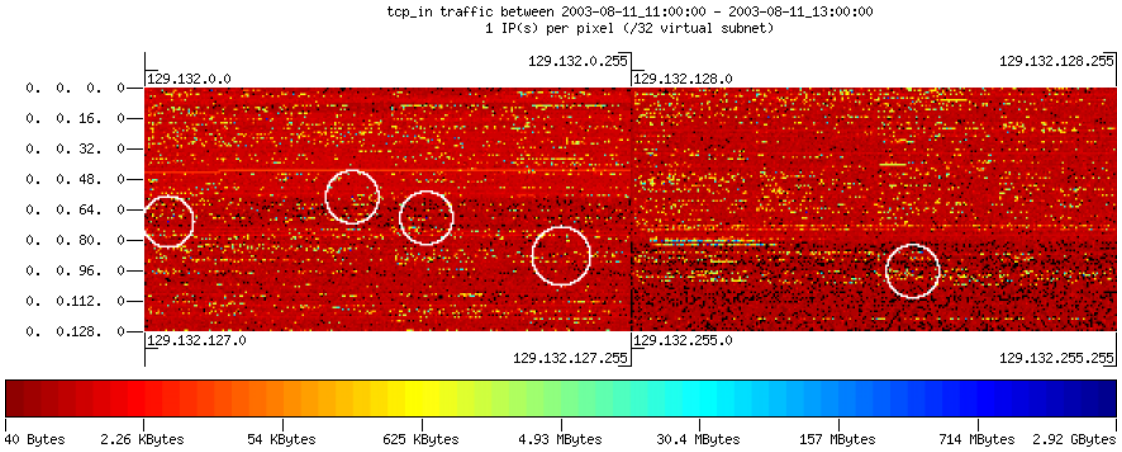

Figure 6.7: IP-Activity (TCP-In) on a normal work-day

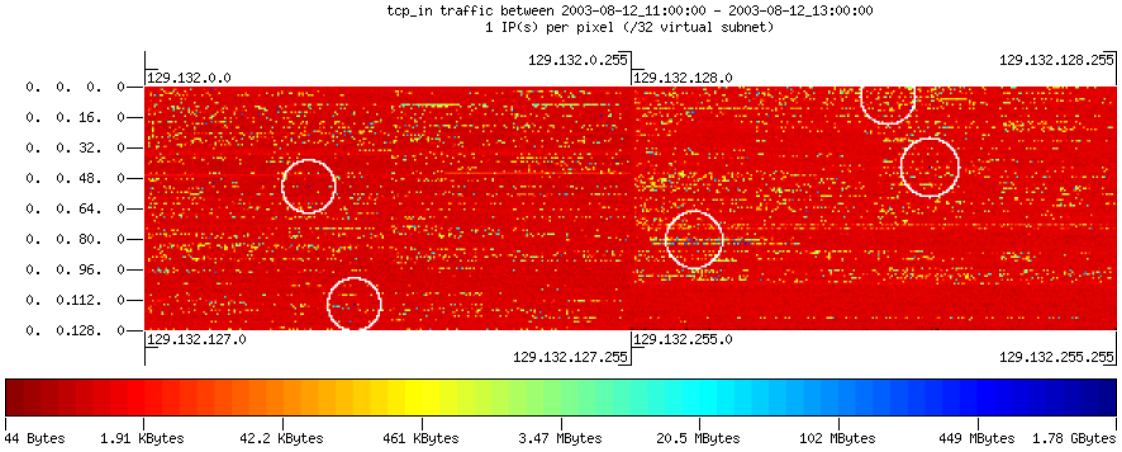

Figure 6.8: IP-Activity (TCP-In) on a work-day for W32-Blaster

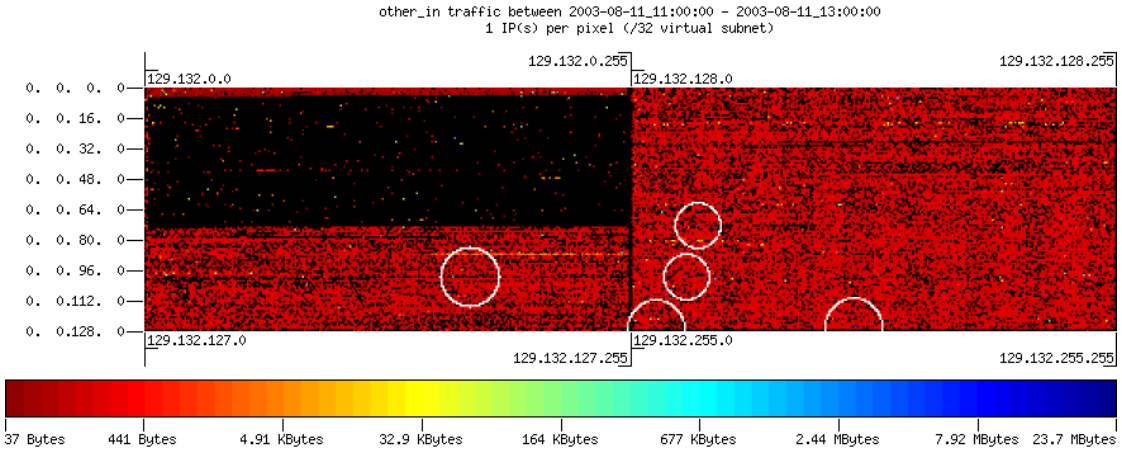

Figure 6.9: IP-Activity (Other-In) on a normal work-day

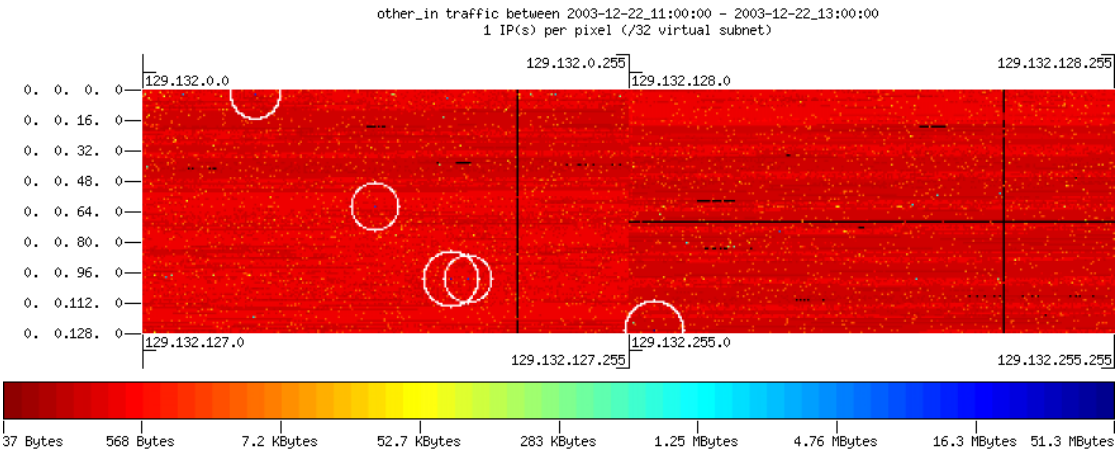

Figure 6.10: IP-Activity (Other-In) on a work-day for Nachi

6.2 Evaluation

6.3 Outlook

During this thesis several ideas for further extensions came up. For most of them there was no time in the scope of a semester thesis to realise them.

6.3.1 3D Flow-Size plot

To give some idea of the variation of the Flow-Size distribution over time, a 3D Flow-Size plot is proposed. As an addition to the implemented Flow-Size plot, we can draw the time also as an axis as shown in Fig. 6.11. The increasing activity of W32-Blaster is clearly observable.

6.3.2 Automatic plot interpretation

The eminent problem of all the presented plots is the necessity to be interpreted by a human network operator. It would be much more handy to let a computer do the interpretation and alert the operator if required.

6.3.3 Characteristic flow size database

During the interpretation of the Flow-Size plots it turned out to be rather difficult to find out what kind of flows/packets are responsible for a peak. One possibility is to look through the databases of the anti-virus companies and try to figure out what kind of traffic some virus produces. A collection with all this insights would be a good starting point for a better traffic understanding.

6.3.4 Offline Netflow data processing

Both plug-ins are designed to work on online data coming in from an UPFrame instance. It would be nice to have some kind of interface to run these plug-ins on the large Netflow archive of the DDoSVax project and create plots of arbitrary time periods.

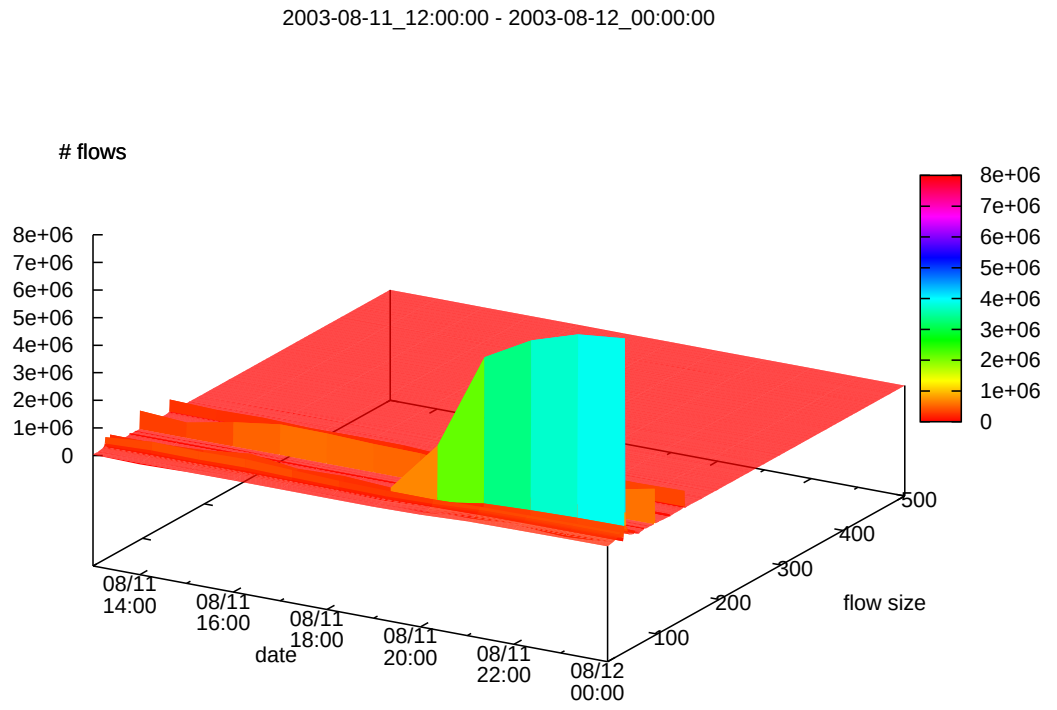

Figure 6.11: Experimental 3D visualisation of a Flow-Size distribution changing over time

6.3.5 Framework to analyse Netflow data on a cluster

To speed up the time consuming process of analysing gigabytes of Netflow data, you can run the job on a cluster. Distributing the work load over all nodes and selecting the right files from the Netflow archive is rather cumbersome to do manually. A scheduler and processing framework for clusters would be handy.

6.4 Acknowledgments

A lot of inputs from my tutor Thomas Dübendorfer and co-tutor Arno Wagner helped to finish this thesis.

I highly appreciated the support by Caspar Schlegel, the author of UPFrame, for responding so fast to my questions and bug reports.

Appendix A

Deployment and Usage

A.1 Building and installation

To successfully run and compile we relay on the following programs and libraries.

Name	Tested Debian packages
ImageMagick	libmagick6_6.0.1.2-1, perlmagick_6.0.1.2-1
gnuplot	gnuplot_3.7.3-3
lzop	lzop_1.01-1, liblzo1_1.08-1
GD Graphics Library	libgd2-xpm_2.0.23-2, libgd-gd2-perl_2.12-1
libbzip2	libbz2-dev_1.0.2-1, libbz2-1.0_1.0.2-1

We also rely on UPFrame [1] and the framework therefore needs to be installed or a compiled copy should reside at `./upframe`. The makefile `./ddos_graphing/Makefile` is used to compile our plug-ins.

To run the CGI scripts you need to copy the following files to the `cgi-bin` directory of your webserver.

File Name	Description
`statReformat`	statistic file re-formatter
`common.pm`	common routines
`create_size_graph.pl`	Flow-Size CGI
`create_ipdist_graph.pl`	IP-Activity CGI
`create_size_graph.cfg`	Flow-Size CGI config file
`create_ipdist_graph.cfg`	IP-Activity CGI config file

In the two config files you have to adjust the path to the statistic files produced by `sizeStat` and `ipDistribution`.

A.2 Command line and CGI parameters

The two programs `create_ipdist_graph.pl` and `create_size_graph.pl` are both dual callable, either as command line version or as CGI program. The different behaviour is automatically selected. For invocation as CGI (HTTP get), the parameters stay the same like for the command line version, just remove any leading `-` or `--`.

Flow-Size plug-in: `sizeStat`	
-h	usage
-o *path*	write statistic files to *path*

-f *filename*	read Netflow data from file *filename*
-p *path*	read Netflow data from hour files at *path*
-s *start*	start with hour file *start*
-e *end*	end with hour file *end*
-u *path*	read Netflow data from UPFrame with FIFO at *path*
-w *command*	use UPFrame's watchdog, restart with *command*

IP-Activity plug-in: `ipDistribution`	
-h	usage
-o *path*	write statistic files to *path*
-f *filename*	read Netflow data from file *filename*
-p *path*	read Netflow data from hour files at *path*
-s *start*	start with hour file *start*
-e *end*	end with hour file *end*
-u *path*	read Netflow data from UPFrame with FIFO at *path*
-w *command*	use UPFrame's watchdog, restart with *command*

Flow-Size CGI/command line: `create_size_graph.pl`	
-h, --help	usage
--sT, --startTime *timestring*[1]	start time of plot
--eT, --endTime *timestring*[1]	end time of plot
-o, --outfile *filename*[2]	output to file *filename*
action [image | form | image_and_form][3]	image: return image only form: return HTML form to enter parameters image_and_form: return HTML form with plot above
--tc [tcp_in | tcp_out | udp_in | udp_out | other_in | other_out]	traffic class
--xMax *n*	plot flows with size less than *n*
--xMaxAcc	accumulate not plotted flows on xMax
--yThreshold *n*	plot flows with count larger than *n*
--xLogscale	logscaled x-axis
--yLogscale	logscaled y-axis
--anim	animation with 60 second frames

IP-Activity CGI/command line: `create_ipdist_graph.pl`	
-h, --help	usage
--sT, --startTime *timestring*[1]	start time of plot
--eT, --endTime *timestring*[1]	end time of plot
-o, --outfile *filename*[2]	output to file *filename*
action [image | form | image_and_form][3]	image: return image only form: return HTML form to enter parameters image_and_form: return HTML form with plot above
--tc [tcp_in | tcp_out | udp_in | udp_out | other_in | other_out]	traffic class
--startIP *ip_address_dotted_decimal*	start IP address of plot
--endIP *ip_address_dotted_decimal*	end IP address of plot
--nGroupBits *n*	group *n* least significant bits together in one virtual IP address
--nStripes *n*	draw plot with *n* vertical stripes

--stripeWidth *width*	*width* of vertical stripes
--plotWidth *width*	plot *width* (just the activity map, excluding all caption elements)
--showPeaks	surround peaks with white circle

A.3 Statistic file format

Lines starting with # denote comments and are ignored. Every line is one aggregation period and follows the syntax described in Listing A.1 and A.2. An example of a Flow-Size and a IP-Activity statistic file is shown in Listing A.3 and A.4.

```
LINE := START_TIME ": " ([SIZE_PAIR] | SIZE_PAIR {", " SIZE_PAIR})

# The aggregation start time (epoch time)
START_TIME := integer

SIZE_PAIR := FLOW_SIZE ":" NUMBER_OF_FLOWS
FLOW_SIZE := integer
NUMBER_OF_FLOWS := integer
```

Listing A.1: Flow-Size statistic file EBNF

```
LINE := START_TIME ": " ([IP_PAIR] | IP_PAIR {", " IP_PAIR})

# The aggregation start time (epoch time)
START_TIME := integer

IP_PAIR := IP_ADDRESS ":" BYTES_TRANSFERRED
IP_ADDRESS := integer
BYTES_TRANSFERRED := integer
```

Listing A.2: IP-Activity statistic file EBNF

```
# Version:   1
#
# startTime: 1060552800
# startTime: 2003-08-11_00:00:00
#
1060553940: 40:4508, 41:612, 43:15, 44:358, 45:3, 46:3, 47:5, 48:2500
1060554000: 40:7867, 41:1042, 43:19, 44:655, 45:4, 46:22, 47:2, 48:4623
1060554060: 40:8089, 41:1114, 43:24, 44:602, 45:5, 46:14, 47:6, 48:4353
```

Listing A.3: Example Flow-Size statistic file

```
# Version:   1
#
# startTime: 1060506000
# startTime: 2003-08-10_11:00:00
#
1060507080: 413927526:1080, 1342235754:1176, 1773853725:168
1060507140: 68787337:840, 73276829:56, 75226774:84, 168922937:240
1060507200: 68362220:56, 56381868:56, 131546072:560, 223036676:1104
```

Listing A.4: Example IP-Activity statistic file

[1] Timeformat: `YYYY-MM-DD_HH:MM:SS`
[2] command line only
[3] CGI only

Appendix B

UML Diagrams

All UML diagrams were generated with Umbrello's [17] re-engineering capability and some hand editing to correct the STL types.

B.1 Base-Class: *aggregator*

The abstract class *aggregator* and its heirs *aggregatorFlowSize* and *aggregatorIPDist* are common abstract data types to hold aggregations over some time period. You can add values (*aggregatorFlowSize::addValue*, *aggregatorIPDist::addValue*) and export the whole aggregation as a string (*aggregator::toString*).

B.2 Base-Class: *netflowConsumer*

Follows the producer/consumer pattern. *netflowFilter* inherits from *netflowConsumer*, filters the incoming Netflow records by traffic classes and forwards them to the corresponding *netflowAnalyser* instance.

The two classes *netflowSizeAnalyser* and *netflowIPDistAnalyser* inherit also from *netflowConsumer* and are responsible for the analysis of the Netflow records.

netflowFileWriter writes the consumed Netflow records to disk and is used in `netflowMerge` to merge multiple Netflow archive files.

B.3 Base-Class: *netflowProcessor*

The producer in our producer/consumer pattern is *netflowProcessor* with its specialisation *netflowUPFrameProcessor* and *netflowFileProcessor*. The first one is the UPFrame reader plug-in

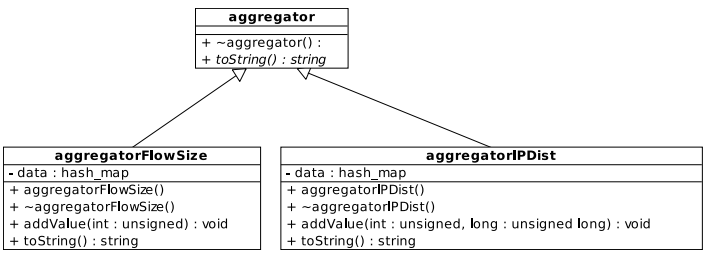

Figure B.1: *aggregator*

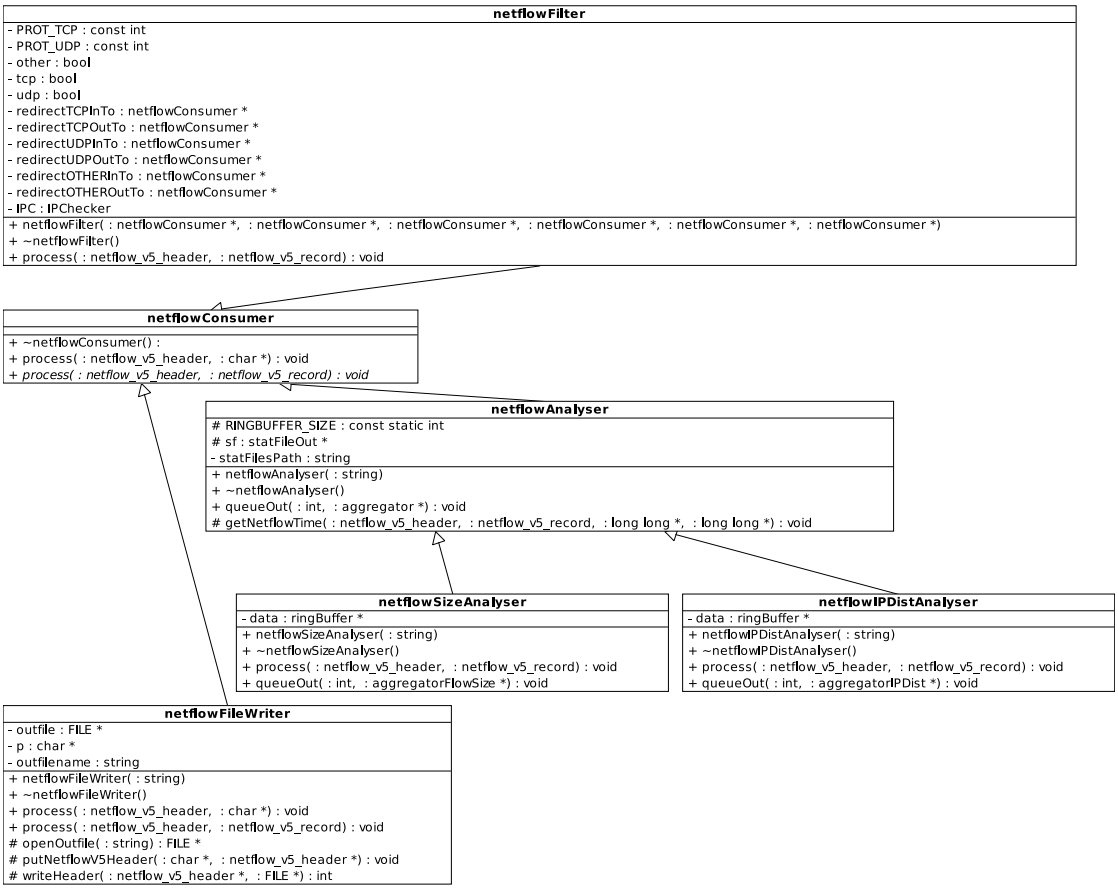

Figure B.2: *netflowConsumer*

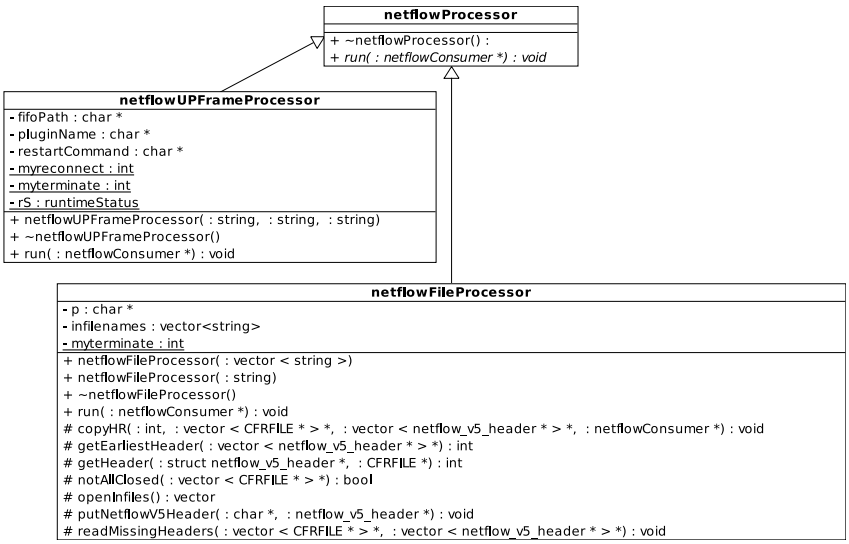

Figure B.3: *netflowProcessor*

and the later reads in one or multiple Netflow archive files. Both classes forward a read Netflow record to the specified *netflowConsumer*.

B.4 Base-Class: *statFile*

The statistic files of both plug-ins are written by *statFileOut*.

The counter part, to read in statistic files, was not implemented. There was no need to process statistic files in the data processing part, which uses this class hierarchy.

We can either write uncompressed text files with *statFileOutTextPlain* or use LZO compression with *statFileOutTextLZO*. There main task is to cache open file handles, select the right file for writing aggregations, provide a mechanism to write file headers (*statFileOutText::writeHeader*) and single aggregations (*statFileOutText::writeLine*).

B.5 Helper classes

In Fig. B.5 we see various helper classes. *ringBuffer* and *queueOutHandler* are two templates. One for queueing aggregations in the described ring buffer and another to implement the queue out operation of the ring buffer to process the overaged and no longer active aggregations.

IPChecker is used to test whether an IP address belongs to the SWITCH network or not.

To encapsulate the library function *scandir* with its callback, *hourFileGetter* is used to search through a directory of Netflow archive files and select the right hour files to process.

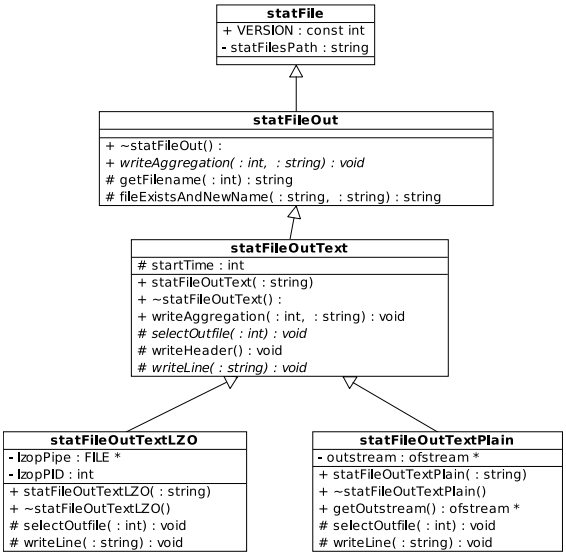

Figure B.4: *statFile*

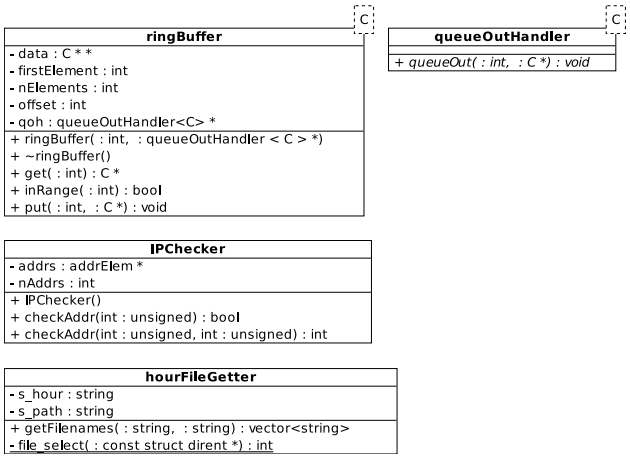

Figure B.5: *ringBuffer, queueOutHandler, IPChecker, hourFileGetter*

Bibliography

[1] Caspar Schlegel: *Real-time UDP Netflow Data Processing Framework*; 2003
`ftp://www.tik.ee.ethz.ch/pub/students/2003-So/DA-2003-26.pdf`

[2] RFC-950: Internet Standard Subnetting Procedure

[3] DDoSVax
`http://www.tik.ee.ethz.ch/~ddosvax`

[4] CAIDA – Cooperative Association for Internet Data Analysis
`http://www.caida.org`

[5] Scylla
`http://www.tik.ee.ethz.ch/~ddosvax/cluster`

[6] System for Internet Level Knowledge (SiLK)
`http://www.cert.org/analysis/silk.html`

[7] cflowd – Traffic Flow Analysis Tool
`http://www.caida.org/tools/measurement/cflowd`

[8] flow-tools – Tool set for working with Netflow data.
`http://www.splintered.net/sw/flow-tools`

[9] nProbe – A NetFlow v5/v9/nFlow Probe for IPv4/v6
`http://www.ntop.org/nProbe.html`

[10] packit – Network Injection and Capture
`http://packit.sourceforge.net`

[11] gri – Programming language for drawing science-style graphs
`http://gri.sourceforge.net`

[12] gnuplot – A portable command-line driven interactive datafile and function plotting utility
`http://www.gnuplot.info`

[13] PLplot – A Scientific Plotting Library
`http://plplot.sourceforge.net`

[14] GD Graphics Library
`http://www.boutell.com/gd`

[15] ImageMagick – Convert, Edit, and Compose Images
`http://www.imagemagick.org`

[16] lzop – Implements the fastest compression and decompression algorithms around
`http://www.lzop.org`

[17] Umbrello UML Modeller – A Unified Modelling Language diagram program for KDE
`http://uml.sourceforge.net`