ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Martin Schreiber

Harpoon: A Flow-level Traffic Generator

Semester Thesis SA, WS 06/07
November 2006 to January 2007

Tutor: Daniela Brauckhoff
Supervisor: Prof. Bernhard Plattner

Abstract

Measuring, analyzing and generating realistic network traffic play an important role for examining network protocols, hardware or applications. Traffic generation tools used to work on packet level, or with specific application/protocol traffic. With Harpoon there exists now a traffic generator on the IP flow level. After a small introduction to the theory of network flows, Harpoon is intensively tested. We focus on the reproduction of recorded traffic. It will be developed a strategy how to rebuild such traffic with harpoon. Harpoon in its initial form has some limitations and weaknesses. It will be shown how they can be solved or avoided. For this purpose new plugins for harpoon were developed, that directly write netflow record instead of produce real traffic.

Zusammenfassung

Das Messen, Analysieren und Generieren von realistischem Netzwerkverkehr sind ein wichtiger Bestandteil zum Überprüfen von Netzwerkprotokollen, Hardware oder Applikationen. Wurde früher auf Packetlevel, oder mit spezifischem Application-/Protokolltraffic gearbeitet, gibt es nun mit Harpoon einen Trafficgenerator auf IP flow level. Nach einer kleinen Einführung in Netflow wird Harpoon ausgiebig getestet. Dabei liegt das Hauptaugenmerk auf der Reproduktion von bereits aufgezeichnetem Traffic. Es wird eine Strategie entwickelt, wie man mit Harpoon diesen Traffic reproduzieren kann. Harpoon besitzt noch einige Limitationen und Schwachstellen. Es wird gezeigt wie diese behoben oder umgangen werden können. Zu diesem Zweck wurden neue Plugins für Harpoon entwickelt, die direkt Netflow Records schreiben anstatt wirklichen Traffic zu erzeugen.

Contents

List of Figures

Chapter 1

Introduction

Harpoon is a flow level traffic generator. It uses some flow characteristics to generate network traffic that can be used for application or protocol testing, or for testing network switching hardware. Harpoons input are IP flow records, recorded at a vantage point in a network, typically routers. It uses this records to extract in a automatic, self configuring step the key aspects of the records, to recreate the traffic.

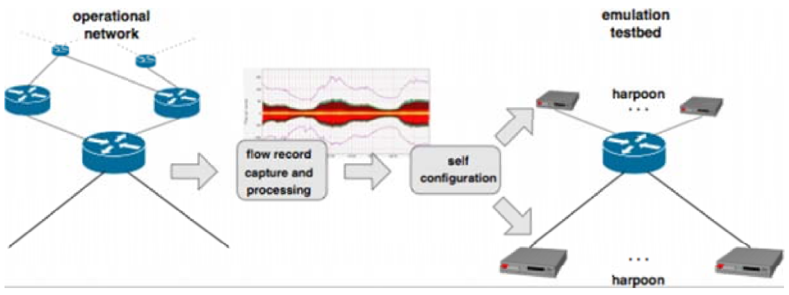


Figure 1.1: IP flow records from a vantage point in a network are used to recreate the traffic

1.1 Motivation

New network protocols or hardware devices, must be tested with some traffic. This traffic can come either from real user interaction or can be automatically generated. The TIK Institute [1] is gathering Netflow data from the border gateway routers from the SWITCH [2] network. These links are heavily used and a huge amount of data (60 mio. flows/hour) are available. The main goal is to use Harpoon, to rebuild that traffic. So that is a future step in this traffic some abnormities can be injected (e.g. caused through a worm) to test automatic detection or other purposes.
To be sure that Harpoon generates traffic with the same characteristics as the input traffic, it gets intensively tested and evaluated. It is important to understand the whole process of generating traffic. So the strength, weaknesses and limitations of Harpoon and traffic generation generally will be understood. There are a lot of challenges, like the scalability or the generation of statistical identical traffic that have to be handled.

1.2 The Task

The first step is to evaluate Harpoon. The focus will be on the scalability and the quality of the generated flows. For scalability, Harpoon is tested with input data from the DDosVax Project [3]. There we will see its behavior under great load. For evaluation the quality of the generated flows, some methods to compare and visualize netflow records have been developed. Because Harpoon has some seriously weaknesses in its scalability, two new plugins as extensions for Harpoon were developed to be able to simulate the creation of the SWITCH data.

1.3 Overview

Chapter 2 is a short introduction to Cisco's Netflow. In Chapter 3 Harpoon is introduced detailed. How it works and especially the automatic self-configuration process. Chapter 4 describes the testbed and tools used to evaluate Harpoon. Chapter 5 finally deals with the evaluation of Harpoon. In Chapter 6 the extensions for Harpoon that were developed were introduced and also tested. Chapter 7 finally is an overview over all tools that were written to work with Netflow records.

Chapter 2

Netflow background

Netflow [5] is a traffic monitoring technology developed by Cisco [4]. As an industry standard a lot of Cisco routers have Netflow implemented as a built-in feature. The advantages for Netflows is its high speed and the less memory usage versus statistics on the packet level. The netflow theory was developed over time so there exists various Netflow versions. Version 9 is the newest one, but the most used is version 5.
Other methods to gather network traffic information are packet level data [6] or SNMP [7]. But Netflows are widely used and today's industrial standard.

2.1 Cisco Netflow v5

A flow, here also named Netflow, is a unidirectional sequence of packets between two endpoints (hosts). So a normal TCP-Connection is comprised of two Netflows (one in each direction). Each flow is uniquely identified by the following seven fields:

- Source IP address
- Source port number
- Destination IP address
- Destination port number
- Protocol type
- Type of service (ToS)
- Router input interface

Differs one of this field of an IP packet, so it is treated as a different flow. The individual flows contain additional information about their start time, the flow duration, number of packets within the flow, byte count within the flow, TCP flags (cumulative OR from all packets in the flow) and some other fields. Figure 2.1 shows a whole Netflow v5 record. The length of every record is 48 bytes.

2.2 Flow aggregation

Usually the Netflows are build in routers within a special cache. From there they are exported over the network as UDP packets [8]. Netflow Collectors receive this packets and save them to disk for further analyze. Such a UDP packet consists of a Netflow header and some Netflow records. To export the flow records in the right time is a hard challenge. There must be a effective cache management. Rules for exporting Netflows are:

- When a Flow is finish (indicated by a FIN or RST flag in the Connection)
- When a flow is inactive for more than an specified timeout

```
 struct NetflowV5Record
{
   unsigned int srcaddr;
   unsigned int dstaddr;
   unsigned int nexthop;
   short inputifid;
   short outputifid;
   unsigned int npkts;
   unsigned int noctets;
   unsigned int flowbegin;
   unsigned int flowend;
   unsigned short srcport;
   unsigned short dstport;
   char pad1;
   char tcp_flags;
   char proto;
   char tos;
   short src_as;
   short dst_as;
   char src_mask;
   char dst_mask;
   short pad2;
};
```

Figure 2.1: Netflow v5 record

- Cache is full

Up to 30 Netflows get collected and exported in one UDP packet. Because the Netflows gets exported with UDP, some packets can be lost. The Netflow header has a sequence number field with the number of exported flow, so that collector knows when some flows are lost. But is is only possible to say how many are lost, it is not possible to recover them.
For more detailed informations, how flows are saved or in detail exported, see [5], or see for a short description and Netflow solutions [9].

Chapter 3

Harpoon

Harpoon [10] is a programm to create network traffic based on the IP flow level. It was developed to satisfy all demands of traffic, not limited e.g. for specific application traffic (e.g. internet telephony over SIP). So it has to cover a lot of different traffic styles. So Harpoon works on the flow level, i.e. TCP or UDP flows, not concerning, what higher protocols or application produced that traffic.

3.1 User behavior concept

The basic idea is to use a user behavior concept behind Harpoon. It mimics users who make file requests and so produce traffic. So also Harpoon is implemented as a client-server structure. The client makes file requests and the server responses. Figure 3.1 shows the basic behavior of Harpoon. Each sessions corresponds to one user, who produces all interconnection times some traffic. Notify that different file transfers can overlap.

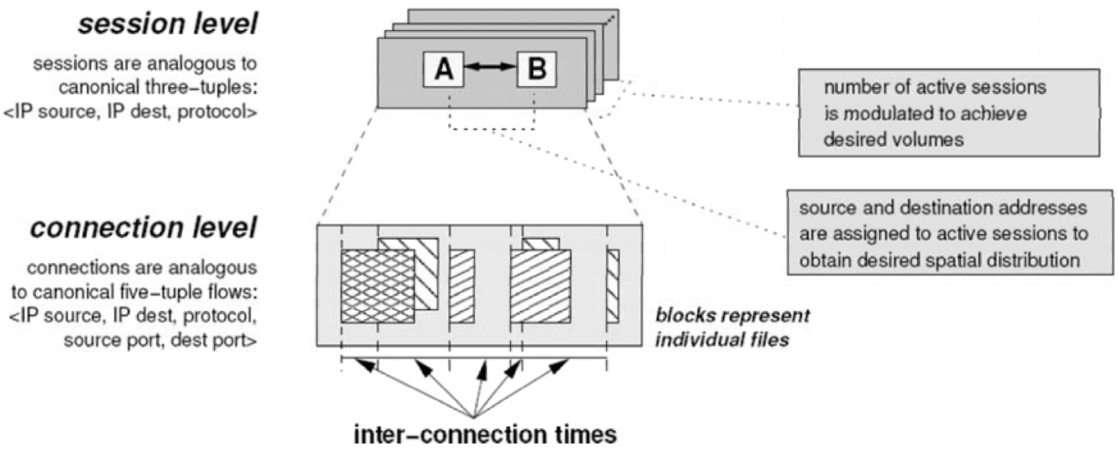

session level
sessions are analogous to canonical three-tuples: <IP source, IP dest, protocol>
A
B
number of active sessions is modulated to achieve desired volumes
source and destination addresses are assigned to active sessions to obtain desired spatial distribution
connection level
connections are analogous to canonical five-tuple flows: <IP source, IP dest, protocol, source port, dest port>
blocks represent individual files
inter-connection times

Figure 3.1: Harpoon's user behavior

These sessions are autonomous, means that harpoon manages only the start of a session and how many sessions are active. What a session finally makes, doesn't matter harpoon. So this sessions are implemented as plugins, which harpoon dynamically loads and executes. There exists different such plugins. One for TCP connections, and some for UDP packets. It is possible to load several plugins and so to create a mixed traffic from TCP and UDP.

3.2 Configuring Harpoon

Harpoon gets as input a XML config file, in which all plugins are written that should be loaded, and for every plugin its parameter. Figure 3.2 shows a sample config file. There are two plugins loaded. TcpClient and TcpServer as written in the *<plugin>* tag. There is also the file name of the plugin code and how many sessions (*maxthreads* attribute) are maximal active (Harpoon will create so many sessions as indicated here). The addresses tags are used to connect to the server. There could also be multiple destination addresses. Then Harpoon would randomly connect to one of the servers. The *interconnection_times* and *active_sessions* tags are the internal distributions described in Figure 3.1. The *file_sizes* tag used by the server are, as the name says, the number of bytes the server will respond to a file request. Details how the TCP plugin works in section 3.4. With this structure various information can be loaded to Harpoon and the plugins, e.g. port number distribution or others.

```
<harpoon_plugins>

  <plugin name="TcpClient" objfile="tcp_plugin.so"
          maxthreads="14" personality="client">

      <interconnection_times> 0.1 0.2 0.3 </interconnection_times>
      <active_sessions> 10 14 6 9 </active_sessions>

      <address_pool name="client_source_pool">
            <address ipv4="127.0.0.1/32" port="0" />
      </address_pool>

      <address_pool name="client_destination_pool">
            <address ipv4="127.0.0.1/32" port="10000" />
      </address_pool>
  </plugin>

  <plugin name="TcpServer" objfile="tcp_plugin.so"
          maxthreads="5" personality="server">
      <file_sizes> 10000 550 900 12000 </file_sizes>
      <active_sessions> 5 </active_sessions>

      <address_pool name="server_pool">
          <address ipv4="127.0.0.1/32" port="10000" />
      </address_pool>
  </plugin>

</harpoon_plugins>
```

Figure 3.2: Sample Harpoon config file

3.3 How Harpoon works

When running Harpoon, there is an internal emulation time. Every session length a new emulation period begins and Harpoon will start all new sessions (there's no overlapping with sessions in different emulation intervals). So every session is active for one emulation period. For each emulation period Harpoon manages how many sessions are active based on the specification in the config file. A client process will iterate once through the *active_sessions* list and then stops. Server processes will cycle through the list. Potentially, every session produces the same kind of traffic. But in each session interval a different number of sessions are active, so the generated traffic will vary.
Harpoons internal parameters:

- Active session distribution: How many sessions in an interval make file transfers
- Interval length: How long a session is

Figure 3.3 shows its behavior. The individual sessions are implemented in different threads, so they are running concurrently. Every session is mapped in a one-to-one fashion to threads. So each threads runs the code from a plugin.

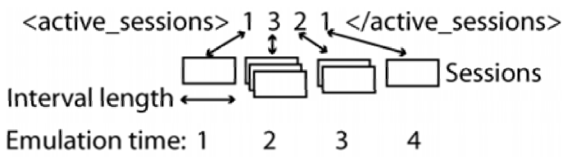


Figure 3.3: Harpoon manages number of active sessions

3.4 TCP Plugin

The TCP plugin is, as the name indicates, responsible for TCP traffic. The plugin has two important distributions:

- File size distribution: How many bytes have to be sent for one flow
- Interconnection times distribution: In what intervals new file transfers are made

This distributions are written in the Harpoon XML config file (see Figure 3.2).
For a session length[1], the client part of the plugin makes all interconnection times a file request to the server. The server will then respond with a file with a random length from the file size distribution.
A Problem is that there can be several active connections and new connection can be requested before an old file transfer ends. So the client plugin has to keep a list with all connections and read from them in a non-blocking way. So several flows can be managed concurrently. Picture 3.1 shows how it works.
A trick the plugin uses, is to reuse TCP connections. Because it has to handle a lot of connections, there could be a problem with port numbers and there is always an overhead with closing and opening a TCP connection. This is a considerable speed problem. So it reuses the connection by sending a TCP_RST (TCP reset) packet at the end of a flow. Then a new flow is established, without the long way of closing and opening a TCP connection.

3.5 Self configuration process

Our main desire is to use Harpoon for regeneration traffic from netflow input. In the DDosVax Project are gathered a huge amount of netflow data. They are gathered from the border gateway routers from the SWITCH [2] network. The goal is to build traffic, that matches the specific characteristics from this gathered traffic.
There are two tools that create the config files for Harpoon from a Netflow file. The first one, a C tool, (harpoon_flowproc) reads a Netflow file and saves it in a ASCII readable format. The second tool, in Python, (harpoon_conf.py) reads this file and creates the final config files for Harpoon. This procedure is illustrated in Figure .

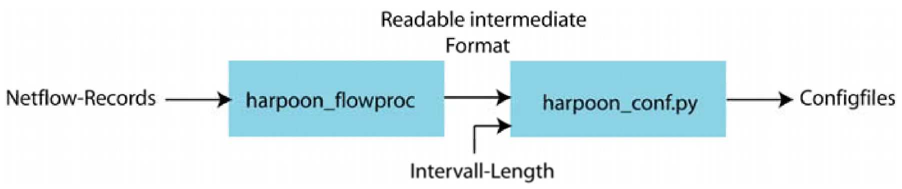


Figure 3.4: Creating config files from Netflow input

Some special attention here have to be turned on the Netflow records. Our records are saved in the network-byte-order. Initially this creation process went totaly wrong, because

[1]session length or interval length are used synonym

harpoon_flowproc used host-byte-order for the flows. So after the first strange results, harpoon_flowproc was edited to read with network-byte-order.

3.5.1 Harpoon_flowproc

This tool reads Netflow records and saves them in a human readable format. Within this step it sorts all flows to the source-, destination address and time. It saves for every source- destination address pair all flows between them in chronological order. Figure 3.5 shows that format on some samples.

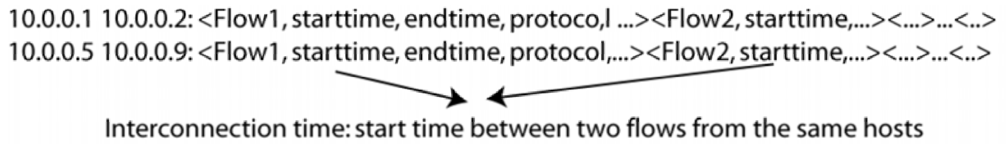

10.0.0.1 10.0.0.2: <Flow1, starttime, endtime, protoco,l ...><Flow2, starttime,...><...>...<..>
10.0.0.5 10.0.0.9: <Flow1, starttime, endtime, protocol,...><Flow2, starttime,...><...>...<..>
Interconnection time: start time between two flows from the same hosts

Figure 3.5: Harpoon_flowproc format for further distribution extraction

It also does some surgery on the flows. Because flows can be exported even when they aren't finished, e.g. when the cache is full or some timeout occurred, one flow can be split up. Harpoon_flowproc can merge these flows, based on TCP flags and start and duration times from the flows. Unfortunately our Netflow records have no TCP flags. So this is needless.

3.5.2 Harpoon_conf.py

This is the tool that does the main part. It reads the file created from harpoon_flowproc and creates the config files for Harpoon, that should create the desired traffic. As mentioned above the TCP plugin has three main statistics: the file size distribution, the interconnection times and the active sessions.
To get the files size distribution, it just takes the flow size from all flows. Likewise easy it can get the interconnection times. The flows are already sorted to the source and destination addresses, so it can take the differences from the start point from two neighbored flows with the same source and destination address (see Figure 3.5).
A bit more tricky is to get the active sessions distribution. But on the way to get this distribution, its easy to get a deeper understanding how harpoon works. To get the active sessions distribution, harpoon_conf.py will mimic Harpoon and so determine how many sessions have to be active to get a specific byte volume.
First of all, it divides the whole flow record in pieces of similar length, the session length. Then, for every interval it determines how many bytes of data are transmitted within this interval. After that, the underlying task of mimic harpoon begins. This is done exactly how the TCP plugin with Harpoon works.
Every interconnection times one file transfer will start. So it takes randomly interconnection times and finds out how many connection attempts are done in the interval. Then its takes for every connection randomly a file size from the file size distribution and sums them up. So it has the totaly bytes that would be transmitted from one session. Are there less bytes transmitted than before determined, it repeats this procedure till it has the desired volume. Then it has the number of active sessions for that period. The Pseudocode 3.6 shows its behavior. Because the active sessions can very vary, depending on the taken interconnection times, the whole procedure is done multiple time and an average value for the active sessions is taken.
Now all distributions are got and the config file with these distributions can be written. As perhaps mentioned, for very large Netflow records it's a bit a greedy thing. Because there are only a lot of samples of the distribution, the config files get fast blown up.

3.6 Problems

There also exists some UDP Plugins for creating UDP traffic. They can either produce a constant bitrate stream, a fixed-interval periodic ping-pong or an exponentially distributed ping-pong

```
#Begin of getting all active_sessions

  for all sessions{
    for some rounds to get an average{
      simulation time t = session_start
      variable active_sessions = 1
      variable bytes_send = 0;
      while true{
        while t < session length{
            t += random inteconnection time
            bytes_send += random file size
        }
        if bytes_send >= bytes_pro_sesson from this session
          goto label1
        else
          active_sessions += 1
      }
      label1:
      remember active_sessions
    }
    do the averaging from all active_sessions to get number of
    active sessions
  }
```

Figure 3.6: Pseudo code for extracting active session distribution

stream. Harpoon tries here to mimic some special traffic. The latter two mimics the NTF (Network time protocol), respectively the DNS (domain name service) protocol. So with the UDP plugins Harpoon drifts away from the idea to create universal traffic to get along with different traffic characteristics. The TCP plugins were implemented to satisfy all different kind of traffic, but the UDP plugins are not so universally implemented. In [10] there's written "... [Harpoon] uses a set of distributional parameters that can be automatically extracted from Netflow traces to generate flows that exhibit the same statistical qualities present in measured Internet traces,...". But this the automatic parameter extraction is only available for the TCP plugin. UDP traffic is just ignored.

Chapter 4

Evaluation Environment

Harpoon gets tested on the Scylla [12] Cluster. There will be two hosts running, one as the server process and the other as the client process.

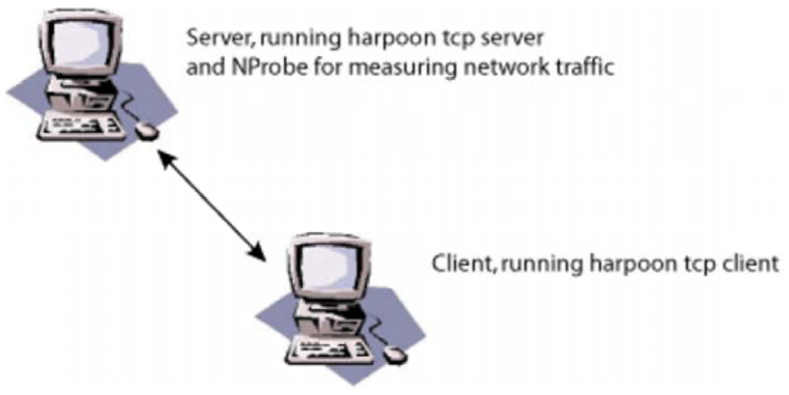

Server, running harpoon tcp server
and NProbe for measuring network traffic
Client, running harpoon tcp client

Figure 4.1: Testing environment for Harpoon

There i used two nodes to generate traffic between them. On the server node, there was also running NProbe (see section 4.2), for listening on the network interface and creating and storing the netflow files. Because there can be some more traffic on this nodes (e.g. the ssh traffic from login, or some other traffic) every captured netflow file contains some foreign traffic. This is unavoidable so i filtered the netflow files for a specific source IP, destination IP and port. So only the generated traffic from harpoon remains. This filtering is done by a self developed tool: netflow_filter (see Chapter 7).
The generated traffic will be analyzed by its intern and extern characteristics. The extern characteristics are the number of flows generated globally and the totaly amount of data sent. The intern characteristics will be the file sizes of the individual flows. We can't have a look at the interconnection times, because all information about the initial source-destination IP information is lost, since we have only two hosts.
In [11] Harpoon got also tested. But we here have different demands. There a config file for harpoon was created with a specific file size distribution and interconnection time distribution. After that, the created traffic generated from harpoon was analyzed if it has this specific characteristics from the configuration. They showed that this worked very well. But we have a netflow record, create from it the config files and we want, that the generated traffic adapts the initial traffic. So especially the config file creation process is important to get the right parameters extracted.

4.1 QQ-Plot

For comparing the initial and generated file size distribution, i used cumulative density function and a QQ-plot [13]. With a QQ-plot diagramm two data records can be compared to their probability distribution. This is done via the p_i-Quantiles. A p_i-Quantil is defined as the smallest x for which $F(x) >= q, 0 < q \leq 1$. This p_i-Quantiles from two distributions will be plotted in one diagram. There will be a linear plot if the underlaying distributions are the same. No matter if one distribution is shifted or re-scaled.

4.2 NProbe

Usually Netflows are measured by routers. Because Harpoon generates network traffic, the traffic somehow have to be measured. But on the Scylla Cluster, there are no routers between the links. My desire is to get all network traffic as Netflow records produced by Harpoon. So i have to listen on a network interface, even more specific on a port on which i will generate traffic.
With NProbe [14] this is nearly possible. It's a tool that can listen on a network interface and export netflows like a Cisco router and that even for fully loaded gigabit ethernet. Even more practical, it can directly dump the Netflow records to disk.

Chapter 5

Evaluation Results

In this chapter we will test harpoon on its ability go generate traffic from a netflow input. First we will test it on a smaller sample of input before we try to generate the traffic from the DDosVax Project.

5.1 Verification of generated traffic from Netflow input

For testing i had to build a reference Netflow file. The goal will be to rebuild this Netflow file with Harpoon. The quality of the generated flows will be evaluated. We will look at the internal and the external characteristics of the generated Netflow records. To get a reference input Netflow file, i used Harpoon with the following configuration:

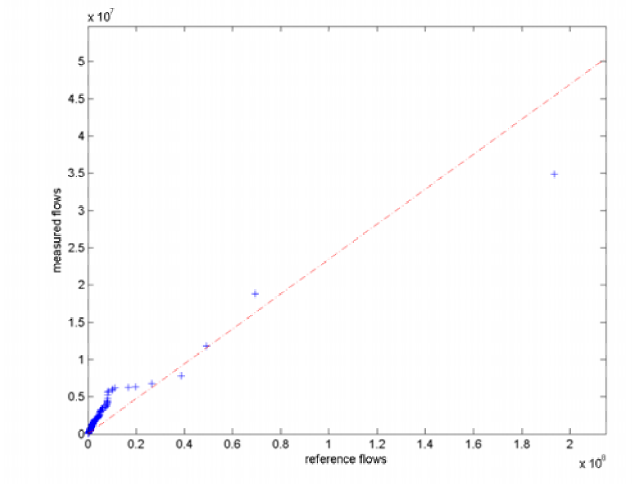
- file size distribution [bytes]: pareto destribution with $\alpha = 1.2$ and $shape = 1500$
- interconnection times [sec]: exponentially distributed with mean and variance one
- record length: 10 minutes
- total flows in record: 11770
- totaly send data: 570MB

With this configuration file Harpoon created traffic that got measured and stored as a netflow record. The first step was to create the new config files for harpoon from this netflow record. In this step you have to decide for a specific session length. To see the impact of this parameter i tested it for several lengthes. Figure 5.1 shows the results. The generation process were three times repeated and the average over flows and sent data was taken. So over an overall look

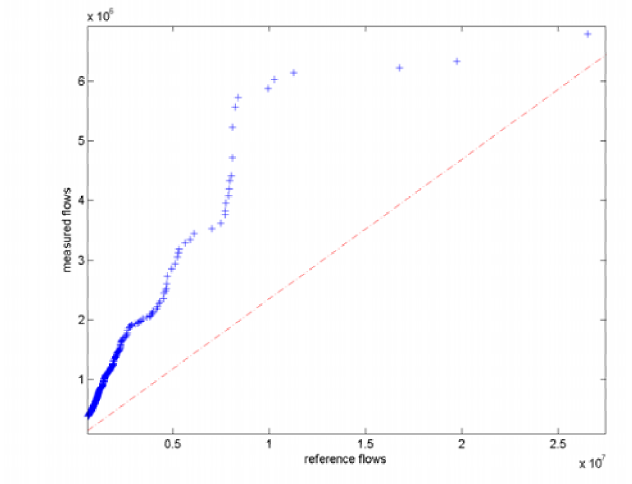
Interval length [sec]	# flows	sent data [MB]
1	32223	1900
10	29751	800
60	33054	1500
300	42560	2000

Figure 5.1: Recreated measured flows from reference input

there are a too many flows and sent bytes produced. For an interval length of 10sec. the overall bytes send is much better than for other interval lengths. But the number of flows is also very bad.
Because the QQ-plot diagrams are very similar for all session length, i show only one here. Figure 5.2 shows the 10 sec session length QQ-plot diagram. The recorded flows don't fit well. Especially for larger file sizes. Some impact may here have the relatively short test for the bigger file sizes. To notion here is, that this is the whole traffic generated from Harpoon. So all client file requests are in there. Because these file requests are very small and always of the same size, they influence the file size distribution. There is no linear plot, so the underling file size distributions from the Netflow record are not the same. Also there are too much flows and data generated. So the internal and the external characteristics of these flows don't match.



(a) Over all file sizes



(b) Zoomed in to smaller file sizes

Figure 5.2: QQ-plot from the file size distributions from the input Netflow records

5.2 Scalability test

Harpoon will be tested with the measured netflow data from the SWITCH network. This will show how scalable harpoon is, because this records are huge. I took one hour of data with about 20 millions of flows. In rush hours there can be about 60 million flows per hour. From this 20 million flows there are 7.5 million of TCP flows.
Now Harpoon comes to it's limitations. The config file creation works fine. But already by looking at the created config files can be seen that this won't work. Because Harpoon has this user behavior it needs to simulate a lot of users that are mapped one-to-one on threads. So in some sessions there should be some ten- to hundred thousands of threads active. This cannot be handled, even when using more then one client. A node on the Scylla cluster can handle about 400 threads. So the generation of this traffic is not feasible.

Chapter 6

Netflow Plugins

Because Harpoons TCP plugin isn't very scalable, a new plugin for Harpoon was developed. It should avoid the traffic generation limitations from the threads. This should be done by writing directly Netflows instead of creating real traffic.

6.1 Netflow TCP Plugin

Because we now produce directly Netflows, we can do this in an off-line manner. So the idea is to simulate all threads (sessions) sequentially. This is done by justify that exactly one thread is active, that does the work for all threads that should be active in a session. This thread now produces every interconnection time a new flow until the end of the session. This is as before except that no file request is made but a Netflow record is written. This has to be repeated for every thread that should be active. So it has to keep track for the right start times of the flows which is done according to the active session time.
Because we are now not anymore limited to server-client environments we can also consider spatial IP address distribution as well as port number distribution. So the new plugin has the following distributions:

- File size: For Netflow sizes
- Interconnection times: Start point of new Netflows
- Active threads in session: How many threads would be active, so how many times a session length Netflows are generated
- Source IP distribution
- Destination IP distribution
- Source port distribution
- Destination port distribution

As for all plugins its configuration is generated by harpoon_flowproc and harpoon_conf.py. So there have to be done some change to create the new config files properly.

6.2 Evaluating the Netflow TCP Plugin

The Goal is to rebuild a Netflow file so that it prevents its characteristics. Harpoon uses as characteristics, the file size distribution and the interconnection times. We will also have a look at the overall data generated and the number of flows generated. Before testing the Netflow plugin with the Netflow records from the SWITCH network, i will test it with a smaller Netflow input. We take the same reference flow as before for the tcp plugin.

6.2.1 Evaluation Results

In Figure 6.1(a) and 6.1(b) we see compared to the input Netflow the file size distribution and the interconnection times distribution. Because we have now the IP information in the generated Netfow record we can extract the new interconnection times. We see that the file size distribution is more accurate than with the TCP plugin. Also the interconnection times distribution matches very well. They are more accurate for larger session lengths. This is because in the original Netflow file are large interconnection times. In the Netflow plugin a interconnection time can only be larger than a sessions length, when two sessions have the same source and destination hosts. Because this is not very probably, most of the generated interconnection times are shorter than a session length. We see in the CDF plot of the interconnection times in Figure 6.1(c) that the two graphs are nearly identical, only shifted vertically. That is actual the probability for having interconnection times larger than the session interval. For larger session length, this effect will be reduced.
The internal characteristics of this new Netflow file are the same. The only problem is here, that the overall traffic generated is, like the traffic produced with the tcp plugin, too large. For the huge Netflow records we want to generate it's useless when the overall traffic is a factor three to four wrong. So we have to find methods to do that better. The generated traffic is so bad, that is makes from some hundreds MB of input Netflow traffic more than 2 GB of traffic. The input flow had about 7.5 millions of flows. Because the Netflow plugin writes everything in one file, the maximum file size is soon reached and so no more flows could be generated. But until that state it generated more than 40 million flows. And this even not simulated the whole Netflow. But there are obviously too many flows created, so a research in detail is omitted. We will have a look at the config files and the creation process to fix that.

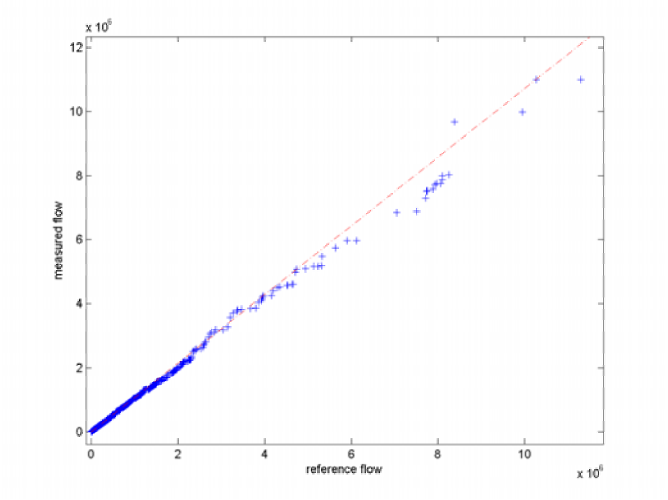
6.3 New automatic config file creation process

The main cause how many traffic is generated lies in the number of active threads per session[1]. There we can adjust how many traffic is generated. In 3.5 we saw how the number of active sessions are ascertained. This is done over the transmitted bytes per session, so why is isn't more accurate isn't clear. Another method we could find the right number of sessions could be done with the number of flows. When there are in the mean the same number of flows, then also the same number of bytes with these flows should be sent. To verify this we can make to following consideration: When we have the same number of flows in a session, we also have the same number of flows for the whole Netflow record. Because they have the same file size distribution, as shown above, the overall traffic should be the same.
There could also be some other approaches. Also possible would be to look at packet level. But this is perhaps not very accurate, because in a Netflow we don't have detailed packet information. A mix of all possibilities would also be possible. I have implemented the version with the flow count.
I have tested the version with the flow count. Figure 3.6 shows the old creation process for active sessions. Now the number of bytes got replaced by the number of flows. To create the config file i again had to change harpoon_conf.py. Because this creation process is unnecessary complicated and slow, i implemented a version of harpoon_flowproc who does the whole config creation stuff. It integrates harpoon_conf.py and so it gets the different distributions and writes the config file. The advantage of this is a huge performance boost. Because in harpoon_flowproc the flows are in a compact format, but harpoon_conf.py deals with very large strings of the same information. Also the part to save and reload the immediate file is omitted. So the config file creation is done a factor 4-5 times faster.

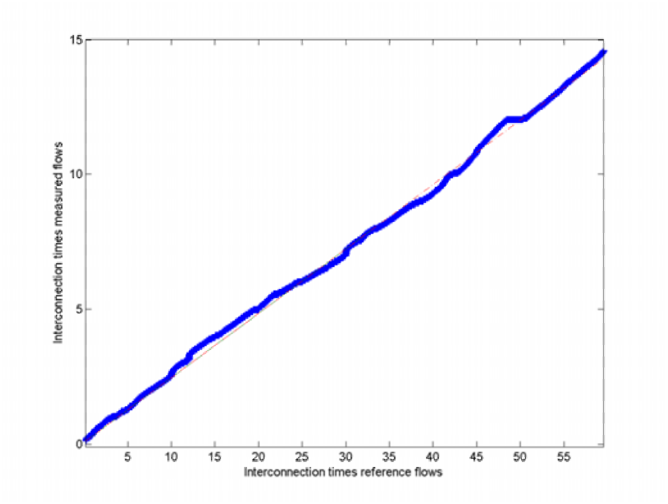
6.3.1 Evaluation of new methods for creating config files

Because we don't have a good match of overall traffic volume and flow counts we have try different approaches to obtain the number of active sessions per interval. With the Netflow plugin we now have a fast tool to create the traffic. Even large Netflows files as input can

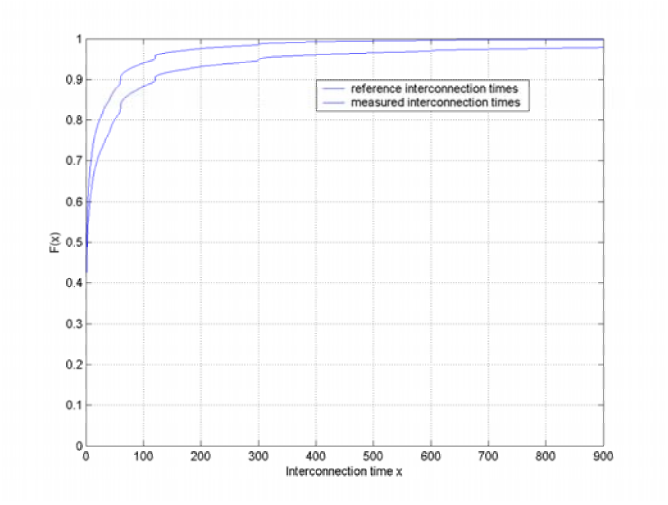
[1] I use the term session as the length of a interval in that the whole record is divided, but also a sessions as a thread in Harpoon that creates for an interval length traffic



(a) QQ-Plot: File size distribution



(b) QQ-Plot: Interconnection time distribution for an interval length



(c) Cumulative density function interconnection times

Figure 6.1: Traffic generated with the Netflow plugin compared to to input traffic

be simulated in some minutes. Because its behavior is very similar to the tcp plugin, when once obtained the right configuration, it can also be applied to the tcp plugin to create real traffic.

Netflow file	# flows	Total bytes sent	Interval length
Input flow	7631778	$1.986 \cdot 10^{11}$	
Plugin output 1	7628810	$1.870 \cdot 10^{11}$	60
Plugin output 2	7632556	$1.876 \cdot 10^{11}$	60
Plugin output 3	7631419	$1.980 \cdot 10^{11}$	900
Plugin output 4	7636323	$1.921 \cdot 10^{11}$	900

Figure 6.2: Overall statistics from created traffic with Harpoon for two different interval lengths

We see in Figure 6.2 that now we can simulate from an input Netflow that didn't worked before. The produced flows and data bytes matches the reference flow very well. So the new created traffic matches the internal and external characteristics of the input flow. The number of active sessions are also dramatically decreased. They are still too much threads to run, but 5-10 times less than before.
Figure 6.3 finally shows the cumulative file size distribution for the reference Netflow record as well as for the measured Netflow record. We see that the two distributions are very close and we can say they are the same. As the file sizes, the interconnection times and the overall flow count and byte count matches the ones from the input Netflow records, the generated traffic has the same characteristics.

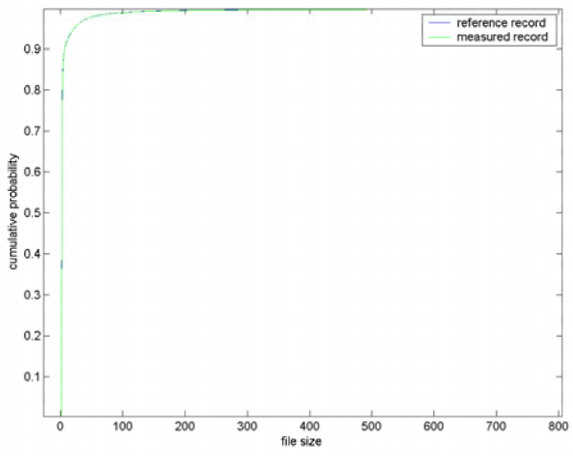


Figure 6.3: Cumulative density function for the file sizes of the tcp flows

6.4 Creating UDP traffic

Till now we have a working plugin that creates TCP connections with the same characteristics as an input record. With the same approach we will also create UDP traffic. Independent of any application/protocol specific traffic.

6.4.1 Netflow UDP Plugin

With the same methods as now derived, i created also a plugin that creates UDP flows. It uses the same approach and the same distributions as the TCP plugin does. So its characteristics are the same as the one from the TCP plugin. Also harpoon_flowproc_conf got changed, so that it now can produce config files for both plugins. With this two plugins, a Netflow stream can be completely rebuild.

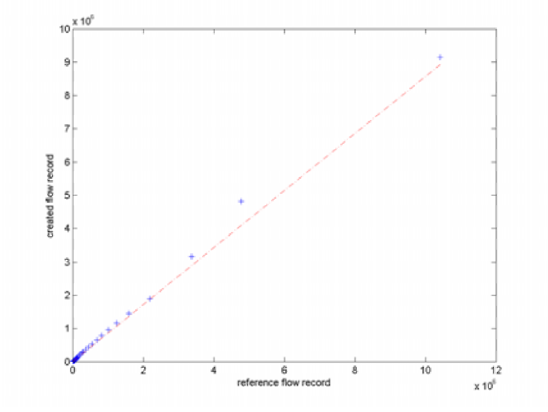


Figure 6.4: QQ-plot of the file size distribution from the UDP plugin

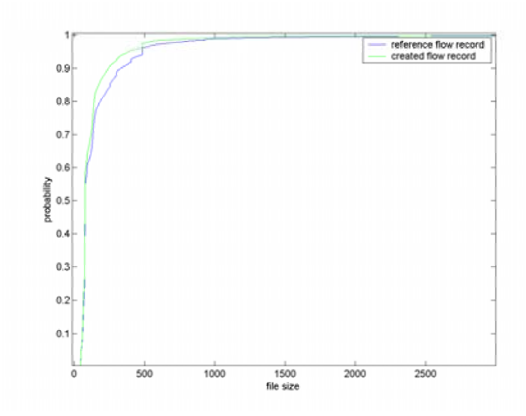


Figure 6.5: Cumulative file size density function from the UDP plugin

Figure 6.5 shows the internal characteristics of the UDP traffic. The file size distribution matches very well to the original Netflow file. In Figure 6.6 shows the complete generation of a Netflow file. Both TCP and UDP traffic are rebuilt. The generation process was repeated three times. We see that is matches the original flow very well, both TCP and UDP.

	Total flows	**TCP flows**	**Total bytes with TCP**	**UDP Flows**	**Total bytes with UDP**
Original flow	23826345	12231431	$3.16 \cdot 10^{11}$	11594914	$1.05 \cdot 10^{10}$
Rebuilt 1	23818081	12227498	$2.84 \cdot 10^{11}$	11590583	$1.42 \cdot 10^{10}$
Rebuilt 2	23830523	12232036	$2.85 \cdot 10^{11}$	11598487	$1.00 \cdot 10^{10}$
Rebuilt 3	23828956	12234866	$2.88 \cdot 10^{11}$	11594090	$1.47 \cdot 10^{10}$

Figure 6.6: Overall characteristics of generated TCP and UDP traffic

Chapter 7

Developed tools to work with Netflows

This work was very affiliate to netflows. Because netflow records are in binary format saved, they aren't very handy. Here i want to present some tools/utilities that were developed during this work to visualize or process netflow files. They are all written in C/C++.

netflow_to_text: A tool that converts netflow records in a ASCII readable format. It takes as standard input a netflow V5 files and produces standard output. It shows IP address pair with port numbers, packets, load and protocol from the flows.

netflow_filter: All the filtering stuff is done with this tool. It takes standard input and produces standard output. It can filter the netflow V5 input file to several rules like source IP address, source port, destination IP address, destination port or protocol.

netflow_statistics: This tool gives some overall statistics about a netflow V5 file. It shows total flow count, flow count individually for TCP/UDP, total byte count as well as byte count for TCP/UDP individual. It also shows the average packets per flow. It takes standard input and produces standard output.

netflow_plot: This tool extracts some information from a netflow V5 record for further processing. So it produces a file with the probability distribution from the file sizes or interconnection times to visualize them with GNUPlot [15]. It can also produce the file size- or interconnection time files for creating the QQ-Plot diagrams or the CDF/PDF diagrams. It takes standard input but depending on the configuration creates the specified files.

nprobe_to_netflow: NProbe just dumps the flows in a file with some additional information. From this a valid netflow V5 file must be created. Thats exactly what this tool does. On standard input it takes all the files created by NProbe and on standard output the valid netflow file is created.

harpoon_flowproc_conf: A newer version of the original harpoon_flowproc. But this version also includes harpoon_conf.py so it directly creates the config files. It also creates config files for the Netflow TCP plugin as also for the Netflow UDP plugin. The created config files have to be embedded in the root XML tag <harpoon_plugins>. It uses the flow count approach to get the number of sessions.

Chapter 8

Summary

Now we have a tool that can generate Netflow records that have the same characteristics, internal as well as external, from an existing Netflow record. The start was with the original Harpoon tool. Already at the beginning we saw that Harpoon wasn't suited for large Netflow records, and that the automatic config file process didn't worked. So a new programm for the creation of config files was developed. For creation of the large flow records other plugins for harpoon were developed, that directly write Netflow record output. Harpoon was limited to create TCP flows. With the same approach as for TCP another plugin was developed that creates UDP traffic. So we have now a tool that can rebuild existing TCP and UDP traffic.
With this tools now some abnormities can be implemented in the config files to simulate worms, viruses or other unwanted behavior for automatic detection or other purposes.

Appendix A

Semester Thesis Task

A.1 Introduction

In large backbone networks where high link speeds are common, security analyses (e.g. anomaly detection) are usually executed on flow data instead of packet data. Flow data (e.g. Cisco Netflow [6]) is faster to process than packet data since information is aggregated. But at the same time, aggregation means loss of information. Consequently, an information for speed trade-off is made here.

In the last years, several anomaly detection algorithms for backbone networks operating on Netflow data [7,3,4] have been developed. However, since even Netflow data is hard or impossible to obtain from network operators (mostly due to privacy concerns) there are no means to study and evaluate these algorithms.

Very recently, a new tool called *Harpoon* [5] has been developed. This tool aims at synthetically generating Netflow data which can then be used for simulations and emulations. *Harpoon* on one hand analyses existing Netflow traffic and extracts feature distributions (e.g., file size distributions) from this data. On the other hand, it uses these emprically determined distributions to generate realistic network traffic.

A.2 The Task

This thesis is conducted at ETH Zurich. The task of this Semester Thesis is twofold. First, the existing *Harpoon* software should be tested and evaluated qualitatively and quantitatively using at least one week of SWITCH [2] Netflow data. Second, *Harpoon* should be extended to allow for plotting the empirically determined traffic feature distributions. Additionally, if time permits, the extended software will be used to perform some initial analyses on intersting flow parameters.

A.2.1 Test and Evaluation of Harpoon

For this task, the Harpoon software first needs to be installed on the DDoSVax [1] cluster. Harpoon is to be evaluated with regard to the scalability of the software (e.g. resource consumption, traffic generation delay), the functionality of the software, as well as the quality of the generated synthetic flows. A methodology for evaluating these aspects will be developed during the thesis.

A.2.2 Visualization Extension for Harpoon

To this date, Harpoon empirically determines certain traffic feature distributions which are used in the traffic generation process. In this task, a plotting extension to Harpoon will be developed which is able to visualize the determined distributions over intervals of different lenght (5 min to 1 week). This task might also require further improvements to the Harpoon tool (e.g. other traffic features).

A.2.3 Optional: Initial Flow Parameter Analysis

If time permits, the developed visualization tool will be used to perform an initial analysis for flow parameter distributions at different temporal (interval length) and spatial (subnets) aggregation intervals.

A.3 Deliverables

The following results are expected:

- *Extensive evaluation of Harpoon.* An evaluation which analyses the scalability, functionality, and flow quality of the Harpoon software is to be performed.
- *A visualization extension for Harpoon.* The Harpoon software is to be extended with a visualization/plotting feature for empirically determined distributions.
- *Documentation.* A concise description of the work conducted in this thesis (task, related work, problem statement, implementation, results and outlook). The Harpoon evaluation as well as the description of the Visualizer extension, and the initial analysis is part of this main documentation. The abstract of the documentation has to be written in both English and German. The original task description is to be put in the appendix of the documentation. One sample of the documentation needs to be delivered at TIK. The whole documentation, as well as the source code, slides of the talk etc., needs to be archived in a printable, respectively executable version on a CDROM, which is to be attached to the printed documentation.

A.4 Organizational Aspects

A.4.1 Documentation and Presentation

A documentation that states the steps conducted, lessons learnt, major results and an outlook on future work and unsolved problems has to be written. The code should be documented well enough such that it can be extended by another developer within reasonable time. At the end of the thesis, a presentation will have to be given at TIK that states the core tasks and results of this thesis. If important new research results are found, a paper might be written as an extract of the thesis and submitted to a computer network and security conference.

A.4.2 Dates

This Semester thesis starts on October 23rd 2006 and is finished on February 2nd 2006. It lasts 14 weeks in total. At the end of the second week Martin has to provide a schedule for the thesis. It will be discussed with the supervisors.
After six weeks Martin should provide a draft of the table of contents (ToC) of the thesis. The ToC suggests that the documentation is written in parallel to the progress of the work.
An intermediate informal presentation for Prof. Plattner and all supervisors will be scheduled 8 weeks into this thesis.
A final presentation at TIK will be scheduled close to the completion date of the thesis. The presentation consists of a 20 minutes talk and reserves 5 minutes for questions. Informal meetings with the supervisors will be announced an organized on demand.

A.4.3 Supervisors

Daniela Brauckhoff, brauckhoff@tik.ee.ethz.ch, +41 44 632 70 50, ETZ G97

Arno Wagner, wagner@tik.ee.ethz.ch, +41 1 632 70 04, ETZ G64.1

References

[1] The DDoSVax project. http://www.tik.ee.ethz.ch/ ddosvax/. SNF no. 20021-102026.
[2] SWITCH - The Swiss Education and Reserach Network. http://www.switch.ch/.
[3] Anukool Lakhina, Mark Crovella, and Christophe Diot.
Diagnosing network-wide traffic anomalies.
In SIGCOMM 04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, pages 219-230
New York, NY, USA, 2004. ACM Press.
[4] Anukool Lakhina, Mark Crovella, and Christophe Diot.
Mining Anomalies Using Traffic Feature Distributions.
In Proceedings of ACM SIGCOMM 2005, August 2005.
[5] Joel Sommers and Paul Barford. Self-configuring network traffic generation.
In Internet Measurement Conference, pages 68-81, 2004.
[6] Cisco Systems Inc. Netflow services and applications - white paper.
[7] Arno Wagner and Bernhard Plattner.
Entropy based worm and anomaly detection in fast IP networks.
In WET ICE 2005, Linköping, Sweden, 2005.

Bibliography

[1] Computer Engineering and Networks Laboratory, ETHZ
http://www.tik.ethz.ch

[2] The Swiss Education & Research Network,
http://www.switch.ch

[3] The DDosVax research project
http://www.tik.ee.ethz.ch/ ddosvax/

[4] Cisco Systems
http://www.cisco.com

[5] Cisco's Netflow White Paper,
http://www.cisco.com/univercd/cc/td/doc/cisintwk/intsolns/netflsol/nfwhite.htm,

[6] Netflow: Information loss or win?
Robin Sommer and AnjaFeldmann, Saarland University, Saarbrücken, Germany

[7] SNMP FAQ,
http://www.snmp.com/FAQs/snmp-faq-part1.txt

[8] NetFlow Export Datagram Format
http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/nfc/nfc_3_0/nfc_ug/nfcform.htm

[9] Cisco's Netflow Introduction
http://cisco.com/en/US/tech/tk812/tsd_technology_support_protocol_home.html

[10] Harpoon: A Flow-level traffic generator
Joel E. Sommers, 2004-2005
http://www.cs.wisc.edu/ jsommers/harpoon/

[11] Harpoon: A Flow-Level Traffic Generator for Router and Network Tests
By Joel Sommers, Hyungsuk Kim, and Paul Barford, University of Wisconsin

[12] TIK Experimental Cluster "Scylla"
http://www.tik.ee.ethz.ch/ ddosvax/cluster/

[13] QQ-Diagramme, eine Einführung,
http://www.geophysik.uni-koeln.de/studium/WS00/vorlesung/geo3/qqplot/qqplot.html

[14] NProbe, An Extensible NetFlow v5/v9/IPFIX GPL Probe for IPv4/v6
http://www.ntop.org/nProbe.html

[15] GNUPlot
http://gnuplot.org