Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Technische Informatik
und Kommunikationsnetze
Computer Engineering and
Networks Laboratory
Communication Systems Group

Emil Haraldsson

DDoS Attack Detection Based on Netflow Logs

Student Thesis SA-2003.35
1st July 2003 / Summer Term 2003

Tutors: Arno Wagner, Thomas Dübendorfer
Supervisor: Prof. Dr. Bernhard Plattner

Table of contents

Table of figures

1 Abstract

This project has been motivated by the growing threat posed by distributed denial of service attacks [1]. To be able to react with short notice to this threat, programs analysing the network data have to be created that could show traffic signatures indicating that a distributed denial of service attack is in preparation or progress. This semester-thesis has used Netflow version 5 [8,15] as the data-format for analysis. The huge amount of data that has to be analysed led to the analysing programs being written in the C programming language with a style trading a bit more memory usage for speed. Programs showing the number of open ports on a single host, the number of bytes transferred, the number of bytes received, package size characteristics and protocol usage have been constructed to provide a firm base for analysis. It has been shown that the implementation of the Netflow version 5 data format has a serious bug that makes it impossible to do certain types of analysis relevant to the detection of distributed denial of service attacks as well as minor optimisation problems, for further information see chapter 6. Proposals of additional statistics that would complete the picture for a more reliable analysis as well as the change to another version of the Netflow data format to make this possible have been made.

2 Introduction

2.1 What a distributed denial of service attack is

A denial of service, DoS, attack is characterized by an explicit attempt by attackers to prevent legitimate users of a service from using that service [1]. This service can be either a specific destination e.g. a host or the medium that connects the host to other hosts e.g. a network. It is an active threat in the sense that an attacker has to actively create these requests in contrast to e.g. traffic analysis and release of message contents [3]. A distributed denial of service attack, often abbreviated DDoS attack, is a denial of service attack, which instead of using one host as the base of attack instead uses multiple hosts, hence the name distributed. There are two basic types of DoS attacks, flooding-based and application-based, the former uses the large number of requests to fill up the victims buffer while the later tries to disable the application in some way making it unusable.

A DDoS or DoS attack differs from an attempt to break into a computer, this is often called a crack or often also falsely referred to as a hack [2], in several ways some of the most prevalent are:

A DDoS or DoS attack doesn't break into the computer attacked, it simply floods the target with so many requests to the service being attacked that it can't handle the legitimate requests coming from legitimate users. There is no break in done during the actual attack, but in the acquiring of a zombies a break in is needed.

A DDoS or DoS attack doesn't in itself provide an attacker with information stored on the target system, it can of course be used as a part of an attack chain of incidents leading to that goal but is in itself not providing such information.

2.2 Why DDoS attacks are a problem

Denial of service attacks can essentially disable your computer or your network. Depending on the nature of your enterprise, this can effectively disable your organization [1]. The main problem with DDoS attacks is that the attack in itself often uses legitimate requests to flood the target, this makes it hard to distinguish from the real legitimate requests that are not part of the attack. This also implies that to distinguish an attacker from a legitimate user we somehow have to be able to verify that the user is a legitimate user and this is time consuming and for services with a lot of users it may as well be infeasible due to the huge overload this would cause on the target system.

2.3 How it is done

There are basically three types of DoS attacks: direct attacks, reflector attacks and unspoofed attacks. In the first one the attacker A spoofs his ip-address, spoofing is the term used for forging ones ip-address to read like someone else‘s [4], to the one of R, and starts sending requests to the victim V. Since the victim V sees that the ip-address in the received packages is the one of R, V sends its replies to R and not to A which originally sent the package, therefore A has not to process the replies from V and can continue at a high rate to send requests to V with R’s ip-address which increases the load on V who has to finish the connection [5]. A doesn’t have to spoof his ip-address to the same host the whole time he can as well continuously change it to read like any other ip-address. The use of randomly generated ip-addresses to use for spoofing has some differences to the described above, the most prevalent is that the randomly generated ip-addresses doesn’t always map to an existing ip-address and hence gives a different answer than if V answered an existing host [5]. This type of attack is used to exhaust the victim’s ability to accept new connections [7].

The second type of DoS attack uses R as a mirror for the attack, this works as follows. A sends a packet to R with the address of V, this can be any type of request that has to give a reply [5], since the source ip-address is the one of V, R sends the reply to V which as well might act according to the standards and perform and further process the data [5]. As above A has not to process the replies form R and can continue at a high rate to send requests to R with V’s ip-address continuously increasing the load on V. A doesn’t have to bound himself to one mirroring address R but can choose this freely. This type of attack is primarily aimed at clogging the victim’s network link [7].

In the unspoofed attack the attacker does not to spoofe his ip-address at all, this has the advantage that the attacker doesn’t have to have root or administrator privileges on the host doing the attack which is the case if spoofed ip-addresses is to be used. This type of attack is not as effective as the two above seen on a single host basis due to that the answers from the victim comes back and has to be processed.

2.4 How multiplicity is achieved and its structure

Using one host as the base of attack wouldn't be enough to bring most systems down, instead the use of multiple hosts in a coordinated strike against the victim is much more effective. To be able to do this the cracker has to take control over multiple hosts. This can be done in a number of ways e.g. cracking the hosts manually, using a virus, which provides the cracker with a backdoor or, by using a worm. In the two later cases the methods of infection are limited to what has been built into the virus or worm and hence if the virus or worm is spreading fast enough this will leave traces in irregularity of traffic on the Internet or any network affected. The type of trace could be an increase in the use of a certain protocol; packet length as well as hosts having a lot of ports open thereby indicating a lot of connections or connection attempts.

When the cracker worm or virus has enough zombies, the term used for a host that is under the control of an attacker or attacking process other terms used are agent and daemons, under its control determined by some criteria either from the cracker himself or by a threshold built in to the malicious code the attack can be started. The attack is started by some means of communication between the cracker and the zombies e.g. an indirect communication as a posting in a newsgroup containing a codeword, a direct communication or by a trigger in the code of the worm or virus that starts the attack.

2.5 The Netflow data format

The data format used for extraction of data was the Cisco proprietary Netflow format. At the time of this semester thesis SWITCH [9] provided me with version 5 of the Netflow format. The latest version being 9, version 5 is a bit aged but widely used together with version 7.

In the Netflow format the term flow is introduced. A flow is identified as a *unidirectional* stream of packets between a given source and destination—both defined by a network-layer ip-address and transport-layer source and destination port numbers. Specifically, a flow is identified as the combination of the following seven key fields:

- Source IP address
- Destination IP address
- Source port number
- Destination port number
- Layer 3 protocol type
- ToS byte
- Input logical interface (ifIndex)

These seven key fields define a unique flow. If a flow has one different field than another flow, then it is considered a new flow. A flow contains other accounting fields (such as the AS number in the Netflow export Version 5 flow format) that depend on the version record format that you configure for export. Flows are processed in a Netflow cache. Netflow version 5 only supports accounting for IP unicast traffic flow. [15].

The following tables describe an exact definition of the parameters used for the statistics the format:

Version 5 Header Format

Bytes	Content	Description
0 to 1	Version	Netflow export format version number (in this case, the number is 5).
2 to 3	Count	Number of flows exported in this packet (1 to 30).
4 to 7	SysUptime	Number of milliseconds since the routing device was last booted.
8 to 11	unix_secs	Number of seconds since 0000 UTC 1970.
12 to 15	unix_nsecs	Number of residual nanoseconds since 0000 UTC 1970.
16 to 19	flow_sequence	Sequence counter of total flows seen.
20	engine_type	Type of flow switching engine.
21	engine_id	ID number of the flow switching engine.
22 to 23	sampling_interval	Sampling mode and the sampling interval information. The first two bits of this field indicates the sampling mode: • 00 = No sampling mode is configured • 01 = 'Packet Interval' sampling mode is configured. (One of every *x* packet is selected and placed in the Netflow cache).

		• 10 = Reserved • 11 = Reserved The remaining 14 bits hold the value of the sampling interval. The sampling interval can have any value in the range of 10 to 16382 (for example, 0x000A to 0x3FFE).

Table 1

Version 5 Flow Record Format

Bytes	Content	Description
0 to 3	Addr	Source IP address.
4 to 7	Dstaddr	Destination IP address.
8 to 11	Nexthop	IP address of the next hop routing device.
12 to 13	Input	SNMP index of the input interface.
14 to 15	Output	SNMP index of the output interface.
16 to 19	DPkts	Packets in the flow.
20 to 23	Doctets	Total number of Layer 3 bytes in the flow's packets.
24 to 27	First	SysUptime at start of flow.
28 to 31	Last	SysUptime at the time the last packet of flow was received.
32 to 33	Port	TCP/UDP source port number or equivalent.
34 to 35	Dstport	TCP/UDP destination port number or equivalent.
36	pad1	Pad 1 is unused (zero) bytes.
37	tcp_flags	Cumulative OR of TCP flags.
38	Prot	IP protocol (for example, 6 = TCP, 17 = UDP).

39	tos	IP ToS.
40 to 41	_as	AS of the source address, either origin or peer.
42 to 43	dst_as	AS of the destination address, either origin or peer.
44	_mask	Source address prefix mask bits.
45	dst_mask	Destination address prefix mask bits.
46 to 47	Pad2	Pad 2 is unused (zero) bytes.

Table 2

3 Problem description

The focus of this semester thesis was to create programs that could be used to identify traffic signatures that with high probability would indicate that a DDoS attack was under preparation or progress.

The task was split into the following subtasks:

3.1 Collection and understanding of traffic raw data

As raw data of the network traffic I used Netflow version 5 [8] traffic logs that were collected recently by a border gateway router of the SWITCH academic network. In a first step the format of this data and its exact meaning was to be understood.

3.2 Definition and calculation of relevant parameters

Relevant parameters for detection of a DDoS attack or a DDoS preparation had to be defined. The parameter extraction could be done using a scripting language such as Perl and could later be incorporated in a future real-time monitoring framework.

3.3 Visualization and interpretation of parameter changes over time

In order to find a DDoS attack a graphical representation of the calculated parameters had to be created. The graphical representation should give the possibility do distinguish between normal and attack data. Signatures that can be applied to identify an ongoing attack or a preparation of the later should be described.

4 Related work

I am not aware of any work being focused on detection of DDoS attacks in backbones or border routers in large networks to this date, accept for the project DDoSVax [21] at ETH [10] under which this semester thesis is written [11]. There is and has been research done in the area of anomaly detection [12] as well as countermeasures [13,14] but not dealing with the same problem as outlined in the problem description of this document as far as I am aware.

5 Design

5.1 Aims

Denial of service attacks or its preparations influence network traffic on many levels, some of the most striking might be: *frequent use of packets having the same length*, this is due to that infectious packets often carry the same payload; *frequent use of the same protocol*, for example the SQL Slammer [16] used udp packets which resulted in an increase in the total use of udp packets over the time of infection; *host or cluster of hosts transmitting more data than usual*, during an attack the zombies transmit large amounts of requests; *host or cluster of hosts receiving more data than usual*, during an attack the attacked host or hosts receive massive amounts of requests; *zombies having large amount of open connections*, during an attack the zombies have a lot of open connections due to the amount of requests they are sending; half open connections, the use of spoofed ip addresses results in a lot of half open connections [5].

I have intended to construct tools providing meaningful statistics that would indicate the traffic signatures above in a way that is graphically pleasing to interpret and that could be used as a base of further reaction. It turned out that it was not possible to construct all the scanners, this is the term I will use for the programs doing the calculations on the raw data and generating the statistics, this was because of an error in the implementation of the Netflow version 5 format that incorrectly always sets the 37th byte in the Netflow flow record format to 0 which should represent an accumulative or of the TCP flags. This resulted in the impossibility to construct scanners that would indicate half open connections that plays a central role in DDoS attacks as explained in 2.3. The importance of the detection of half open connections is due to that over 94% of the DoS attacks use the TCP protocol [22].

The scanners were going to work on large amount of data, each Netflow file averaging 140MB in bz2 [17] compressed format. I decided to construct the scanners using the C programming language and not a scripting language such as Perl which when scanning multiple Netflow files would be too slow to be a realistic implementation used for close to real-time detection of DoS and DDoS attacks. For the decompression and parameter extraction I chose to use a framework for Netflow data files written by Arno Wagner [18] as well as a hashtable implementation written by Arno Wagner [18].

The output of the scanners are written to disk as normal text-files optimised for display by gnuplot [19]. I choose to use gnuplot because it is a widely used graphic representation program and because its ease of configuration that would give further users of my programs the opportunity to fine tune the graphics to their needs.

To run the scanners effectively I wrote a framework in the bash programming language [20] that could easily be modified to run effectively in any linux/unix environment which supports bash commands. To edit the bash program knowledge about the uptime in unixseconds of the Cisco router providing the Netflow data has to be obtained. The program is called as follows:

```
ddosspy {relative or complete path to the netflow data files}
```

All the scanners work on the bz2 compressed formats as well as on decompressed Netflow data files. The data to be analysed can be piped to the scanner as well, this is done in bash as follows:

```
cat {netflow data files} | name-of-scanner {parameters}
```

5.2 Specification of the different scanners

5.2.1 Protocol statistics

The scanner can be configured by the command line to output the statistic files with different resolution in seconds. The scanner is called as follows.

```
protocolscanner -u {unixseconds on router (ms)} -r {system uptime on
hardware engine (ms)} -l {time of first packet in the flow (ms)} -i
{accuracy of scanning in seconds} -f {netflow data file to analyze}
```

eg.

```
protocolscanner -u 1052442445 -r 2372517516 -l 2372518952 -i 60 -f
xxx.dat.bz2
```

More information can be obtained calling: `protocolscanner -h`

5.2.1.1 Motivation

To discover the case of an infection- or attack-phase relying heavily on a single type of protocol I constructed a scanner showing the number of packets per protocol being uploaded and downloaded with respect to a network. The network of interest was the one run by SWITCH which network addresses were coded into the implementation of this scanner. The scanner shows in excess to the number of packets uploaded and downloaded from the SWITCH network the total number of packets being uploaded and downloaded, making it easier to track total traffic flow.

5.2.1.2 Algorithm

It is assumed that the packets are evenly distributed over the time of each flow.
6 predefined arrays of type float:
Upload: out_tcp_packs_array, out_udp_packs_array, out_others_packs array

Download: in_tcp_packs_array, in_udp_packs_array, in_others_packs_array

- Set the default value of the arrays to 0

- Extract user parameters

- For each flow in all the files, if multiple, or the only one if one

1. Extract the number of packets in the flow
2. Extract the type of protocol
3. Extract the start time of the flow
4. Extract the end time of the flow
5. Extract the nexthop ip-address to determine if inward or outward bound flow e.g. in this implementation all flows having its nexhop ip-address starting with 130.59. would be considered inward bound.
6. Compute the number of array elements that are to be increased

Determine which array is going to be edited by using type of protocol and nexthop ip-address

7. Calculate the percentage influence on the first array element and insert

For all array elements but the last

8. Insert the value in the array

For the last element

9. Calculate the percentage influence on the last array element and insert

- Print to the specified output file, default is stdout.

5.2.2 Packet size statistics

The scanners can be configured by the command line to output the statistic files with different resolution in Bytes. The scanner is called as follows.

```
sizescanner –b {Byte size resolution} –f {netflow data file to analyze}
restscanner –b {Byte size resolution} –f {netflow data file to analyze}
```

more information can be obtained calling: `sizescanner –h, restscanner -h`

5.2.2.1 Motivation

In the case of a worm or virus infection the infectious packets often carry the same payload or a limited number of different payloads, this is true for attack packets as well. This gives rise to one or a limited number of frequently used package sizes. To have the opportunity to see if this is the case in the network analysed I have created two types of scanners, the first showing the average packet size in each flow and the other showing the total layer-3 bytes transferred in the flow modulo the MTU, maximum transfer unit. They should indicate if there are packet lengths that are frequently used and could indicate an attack preparation or progress.

5.2.2.2 Algorithm

5.2.2.2.1 Average byte size of packets

One predefined array of type long int.

- Set the default value of the array to 0
- Extract user parameters
- For each flow in all the files, if multiple, or the only one if one

1. Extract the number of packets in the flow
2. Extract the number of transmitted bytes
3. Compute the average byte size of each packet
4. Increase the array element corresponding to the size in step 3 with the number of transmitted packets

- Print to the specified output file, default is stdout.

5.2.2.2.2 Layer-3 Bytes transferred modulo the MTU

One predefined array of type long int.

- Set the default value of the array to 0
- Extract user parameters
- For each flow in all the files, if multiple, or the only one if one

1. Extract the number of packets in the flow
2. Extract the number of transmitted bytes
3. Compute the rest of the transmitted bytes modulo MTU (1500)
4. Increase the array element corresponding to the size in step 3 with one.

- Print to the specified output file, default is stdout.

5.2.3 Statistics of number of connections

The scanner only takes the Netflow data file as input. The scanner is called as follows.

```
portsscanner –f {netflow data file to analyze}
```

More information can be obtained calling: `portsscanner -h`

5.2.3.1 Motivation

In an attack or infection phase the attacking zombies or the computers infecting others commonly do a series of attempts at a time. During the attack phase the goal is to make as many connection attempts as possible and hence the zombies might have a lot of

connections open. This might also be the case during a portscan which results in a similar pattern if connection oriented protocols are being used, therefore it is hard to distinguish a computer infecting others with a worm or virus from a computer doing a portscan, using the scanner explained in 5.2.1 and 5.2.2 might help to distinguish the above while in a portscan there is not as much data generated using a single protocol as in a massive worm spreading and the size of the packets probably vary more than in the case of an infection or DDoS attack. To be able to see what computers have a lot of connections open I have constructed this scanner to pinpoint that issue. Since the amount of data related to each ip address that could occur in this statistics is too big for a linear search I have used two hash tables to store the data. The first hash table is used during scanning and contains the ip address concatenated with a colon and the source port, the second hash table used for presentation is used to reorganize the data for the printout. It consists of each occurring ip address in the former hash table as a key and the number of open ports as the value bound to that key. The reason for using this approach was speed. If I had used only one hash table I would have had to store all the number of used ports as the value corresponding to that ip-address, this means that I would have been forced to search through the values for a corresponding key every time of a new insertion to make sure that the port hadn't already been stored in the value of that key. This searching would in a basic approach be linear and in since there are 65535 possible ports that could appear for each key this would slow the scanner down considerably. There is though an advantage with the later method, if the scanner is to be run on a extensive amount of Netflow data files in one run the method I have implemented which needs more space during runtime than the later could if the machine that is running the scanner runs out of RAM memory have to access the hard drive and could ultimately become slower than if storing all information in one hashtable. It was specified that the scanners were wished to run in as close to real-time as possible in the specification and I have hence constructed the scanner using two hashtables which is faster on a small number of Netflow data files.

5.2.3.2 Algorithm

Two predefined hashtables dynamically growing and shrinking.

- Extract user parameters

- For each flow in all the files, if multiple, or the only one if one

1. Extract the sender ip address
2. Extract the source port
3. Concatenate the ip-address with: {source port number}
4. Insert in the hashtable using the result in stage 3 as key and a null value

- For each entry in the above hashtable

1. Extract the ip-address from the key in the above hashtable
2. Use the ip-address as key in the new hashtable and increase its value by 1

- Print to the specified output file, default is stdout

5.2.4 Layer-3 bytes transmitted statistics

The scanner only takes the Netflow data file as input. The scanner is called as follows.

```
brecscanner –f {netflow data file to analyze}
```

More information can be obtained calling: `brecscanner -h`

5.2.4.1 Motivation

In an attack phase the zombie hosts are sending large number of requests to the attacked computer. In the infection phase of a fast spreading worm the infected hosts send a large number of requests to try to infect other computers. The extensive sending of requests might show off in statistics showing how much data is received on a single host basis showing the address or addresses of the attacked computer or computers. It might as well show that what looked as a distributed denial of service attack really was a flash crowd, the term used when very “hot” data is put on a server available for download by a large number of hosts which then rush to download the new data, which has similar patterns to a DDoS attack i.e. a lot of hosts making requests but with the difference that the computer being queried actually completes the connection and start sending the requested data. I decided to create this scanner to in cooperation with the scanner explained in section 5.2.5 detect flash crowds and possibly attack and infection schemes.

5.2.4.2 Algorithm

One predefined hashtable dynamically growing and shrinking.

- Extract user parameter

- For each flow in all the files, if multiple, or the only one if one
1. Extract the sender ip address
2. Extract the number of layer-3 bytes transferred
3. Insert in the hashtable using the sender ip address as key and increment the value correlating to the key with the number of layer-3 bytes transferred

5.2.5 Layer-3 bytes received statistics

The scanner only takes the Netflow data file as input. The scanner is called as follows.

```
btrascanner –f {netflow data file to analyze}
```

More information can be obtained calling: `btrascanner -h`

5.2.5.1 Motivation

I decided to create this scanner to complete the view of the scanner described in 5.2.4. The two together would show if there are one or more hosts receiving a lot of data and if the same host or hosts are transmitting a lot of data. If the host or hosts were receiving a lot of connections on a short period of time as well as transmitting a lot of data it would probably be a flashcrowd, but if the host or hosts would receive a lot of connections but

send very little data it could be a denial of service attack. It should be pointed out that the server computer would at a total receive a lot of data due to the number of requests even though the request packets are generally small.

5.2.5.2 Algorithm

One predefined hashtable dynamically growing and shrinking.

- Extract user parameter

- For each flow in all the files, if multiple, or the only one if one
4. Extract the receiver ip address
5. Extract the number of layer-3 bytes transferred
6. Insert in the hashtable using the receiver ip address as key and increment the value correlating to the key with the number of layer-3 bytes transferred

6 Results

I have for each constructed scanner found patterns indicating the functionality and relevance of each of the scanners individually. It must be pointed out that each scanner used on its own doesn't provide enough information to act on in a real implementation scenario, used instead in correlation to each other they are much more effective in showing what is really going on which was always the intended way of using the scanners.

I have found that the scanner displaying the average byte size of each packet, 5.2.2, has some patterns that are very often present. It shows a large peak at 40 Bytes as well as at 1500 Bytes, the sooner, which are payload less ACK packets and the later deriving from the fragmentation of packets at the MTU. There are also three more peaks frequently occurring which I have not found an explanation for at 404 Bytes, 551 Bytes as well as at 1064 Bytes.

The two scanners displaying received and transmitted bytes respectively show a very unevenly distributed transmitting and receiving pattern. While some peaks seem to be widely used servers and are continuously transmitting information others are more variable in use depending on time.

The statistics showing the number of open ports per host has shown the frequent use of portscans, which seem to occur all the time and are now not an exception in the Internet traffic but rather a part of it. It has also shown distributed behaviour on small ranges of ip addresses which might be a network infected with a virus, worm or alike, these ip address ranges seem to be prevalent over longer periods of time indicating that the problem is either not detected or not being fixed.

More surprising results were that the implementation of the Netflow version 5 format incorrectly sets the 37:th byte in the Netflow flow record format to 0, this should have

been fixed by now since this version of the Netflow data format still is widely used and that it is vital information especially for this type of anomality detection. The Netflow version 5 format also has minor potential for optimisations as it shows redundant information, the information contained in the version 5 header format byte 12-15 showing the unix nanoseconds since 0000 UTC 1970 already includes the information shown in the 8-11 byte of the version 5 header format showing the unix seconds since 0000 UTC 1970.

6.1 Samples and interpretation

All the graphs below are run on the same Netflow log files coming from a single hardware engine, the time indicated by figure 2.

6.1.1 Protocol statistics

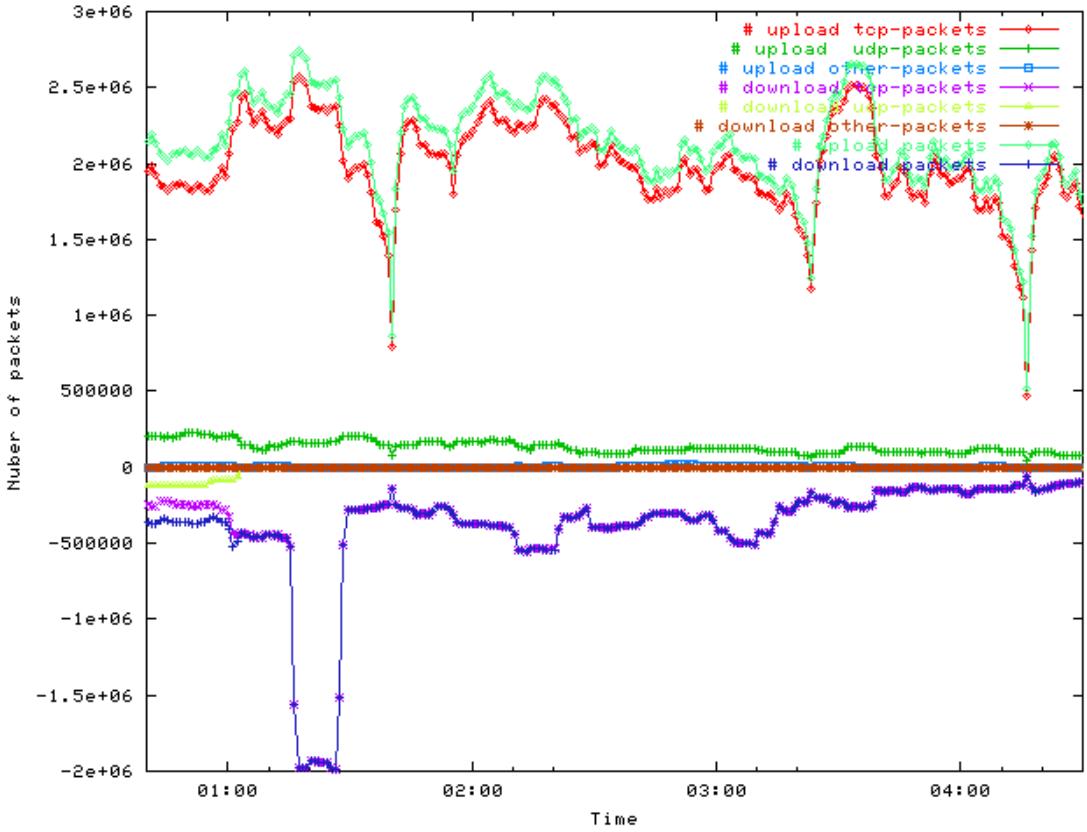

Figure 1

This statistics shows the algorithm explained in 5.2.1 applied on 4 hours of Netflow data in the SWITCH network. We can see that it is mid night in Switzerland mirroring itself in the low download rates in contrast to the relatively high upload rates from the SWITCH network. The major part of the traffic being TCP in both directions. At about 01:20 we can notice a drastic increase in downloaded TCP packets for about 15 minutes, this could be a lot of things e.g. big media files being downloaded or the alike. The upload pattern has some deep ridges, which should be considered normal for Internet traffic.

6.1.2 Packet size statistics

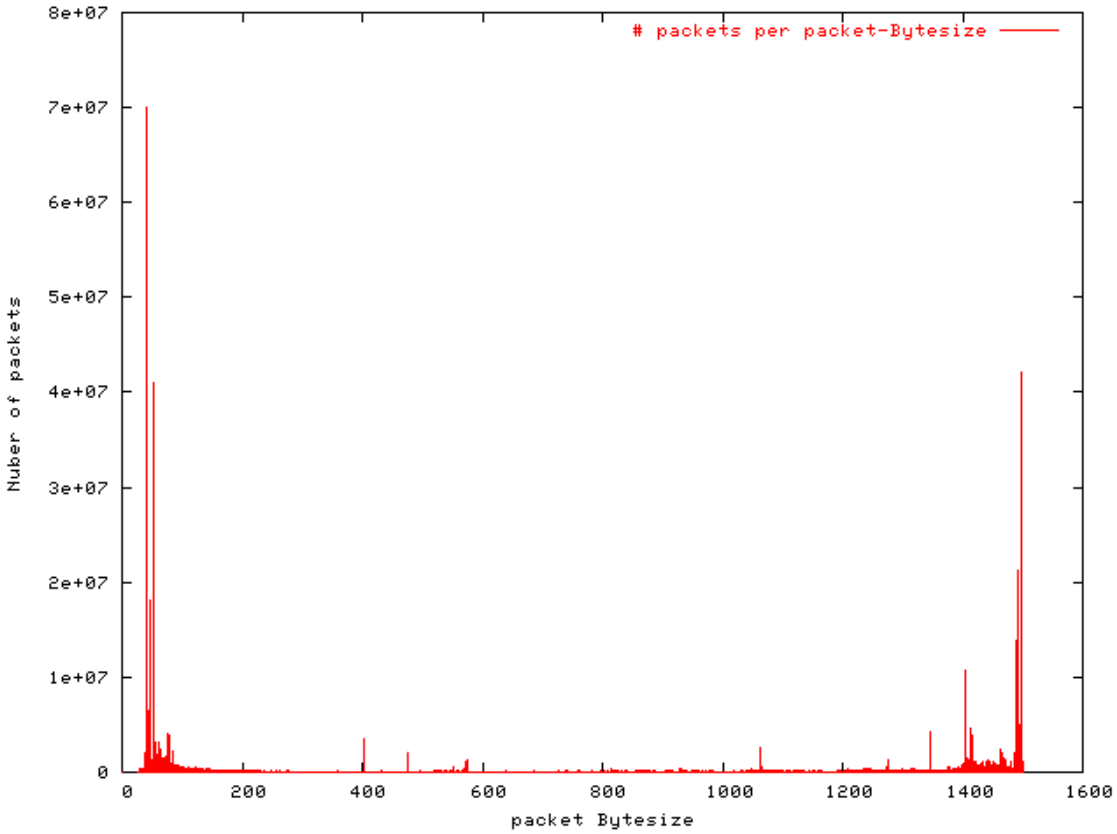

Figure 2

The algorithm used is the one explained in 5.2.2. We can observe the huge peak at 40 Bytes, which are payload less ACK packages as well as the big peak at 1500 that is the MTU and shows the big number of defragmented packages. We have three big peaks at 44 Bytes, 56 Bytes and 1488 Bytes respectively. To be able to tell if the peaks are suspicious or not we would really need to know a lot more of what is normal in this network, which is outside the scope of this thesis. Worms like the recent SQL-Slammer would show up clearly in this statistics generating a peak at 376 Bytes, the specific size of the SQL-Slammer worm, that without any great previous research in normal network behaviour could easily be recognized.

6.1.3 Host's number of connections statistics

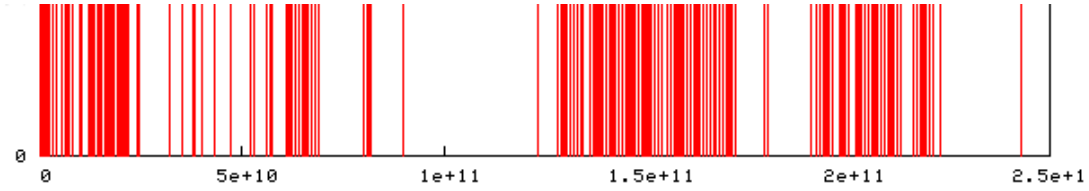

Figure 3

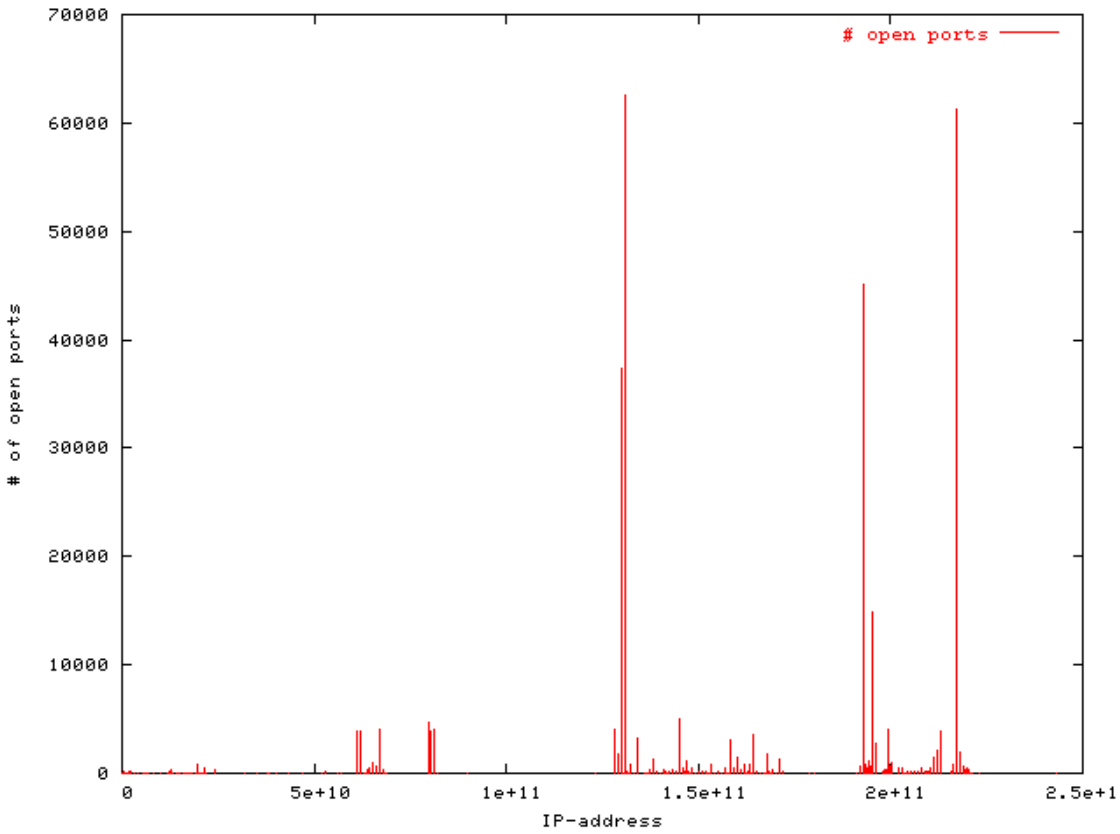

Figure 4

This statistics show the number of open ports for each host generated by the scanner explained in 5.2.3. The first plot in figure 4 shows the number of hosts that are actually connected to the Internet in the sense that they have at least one port open communicating. In figure 5 we can observe four massive peaks two of which have almost all their ports open. It should be pointed out that in this broad view of the traffic the peaks shown could be more than one machine i.e. actually clusters of machines having this many ports open which is shown in the next plot below.

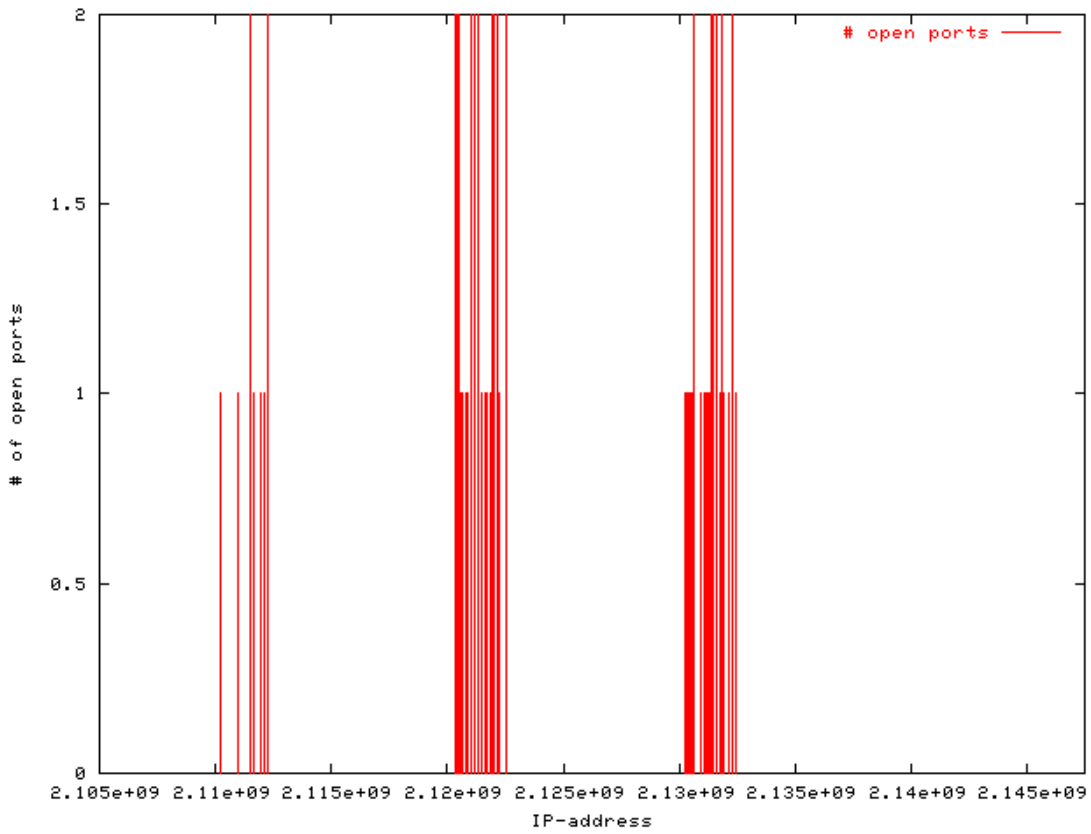

Figure 5

We can see that what looked like a single host in the previous graph actually was a cluster of hosts having multiple ports open, further zooming would show the hosts on a single IP address basis making it possible to distinguish hosts acting as zombies in a DDoS attack or being infected by a worm, virus etc trying to infect other hosts.

6.1.4 Layer-3 bytes transmitted statistics

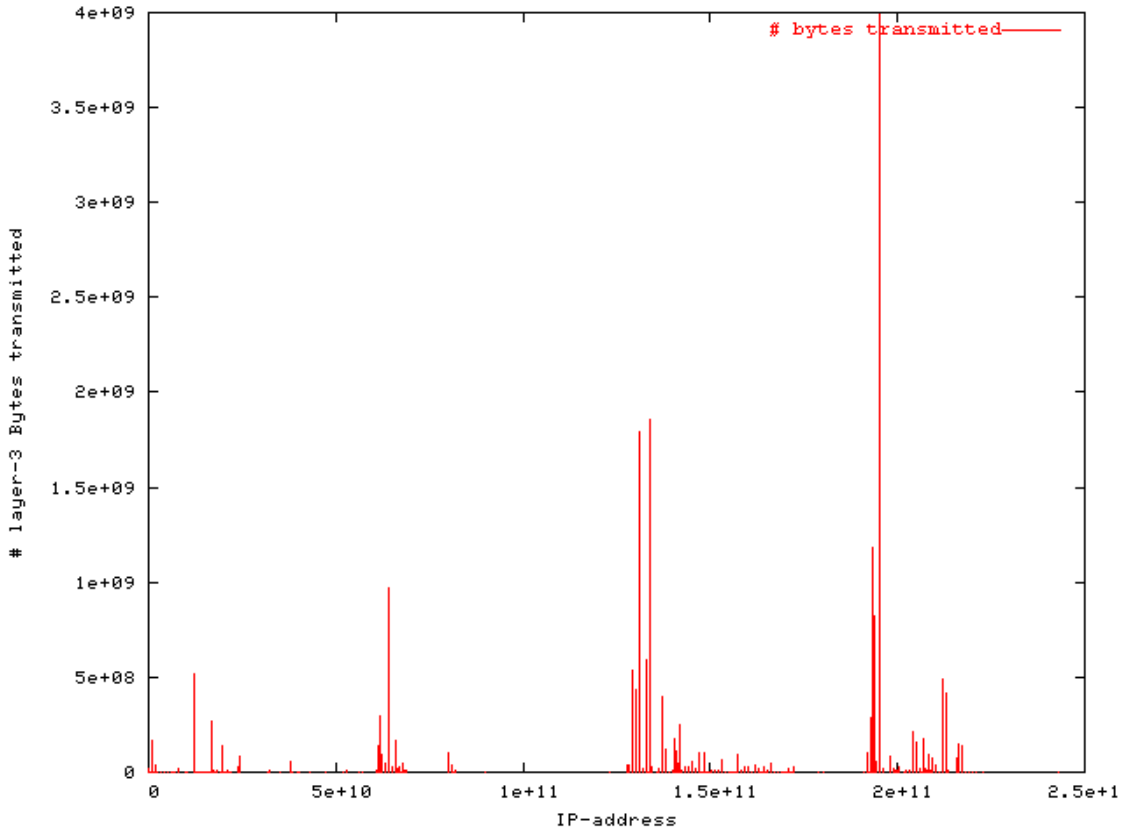

Figure 6

The statistics above shows a big irregularity in data transmission on the network layer. We can se that some of the computers having a lot of ports open also are transmitting a lot of data. This could be a worm or virus spreading as well as a DDoS attack. To know for sure we have to look closer at the single IP-addresses to see the patterns more clearly.

6.1.5 Layer-3 bytes received statistics

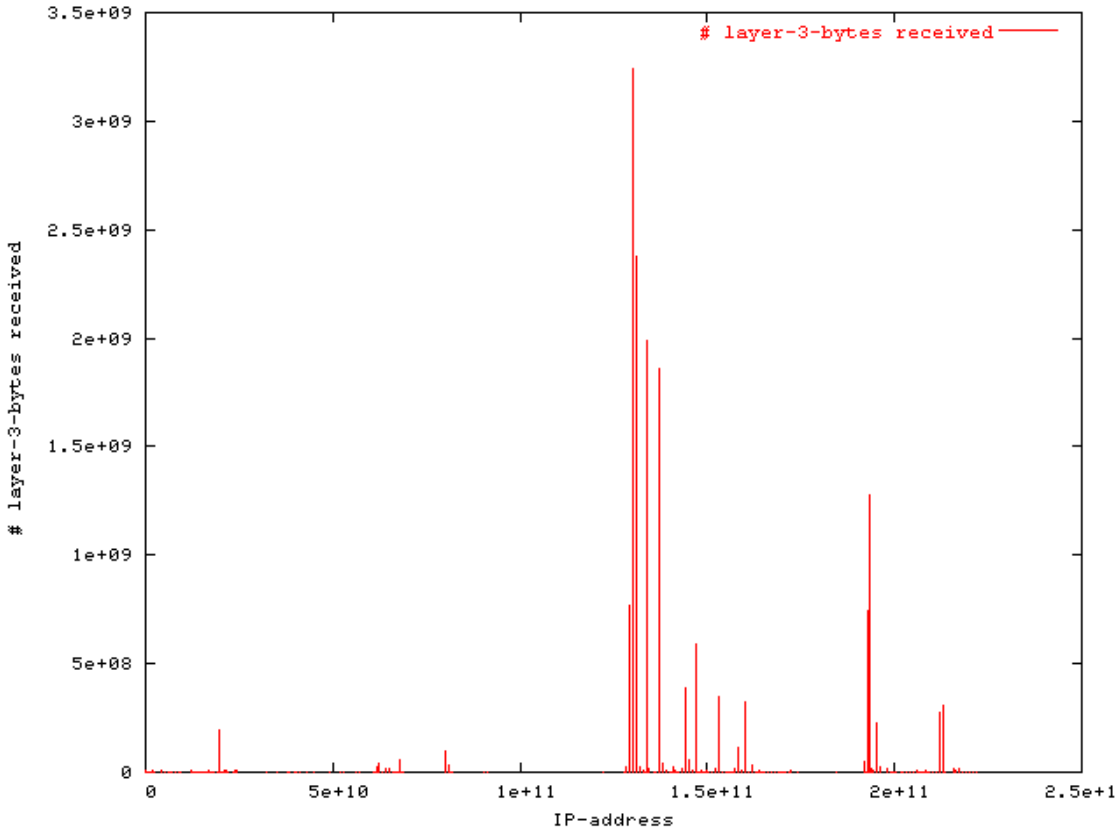

Figure 7

This graph show the number of layer-3 bytes received for each IP-address. We can correlate this graph to the one above and see that the big transmitters of information are not part of the hosts that receive the most data. The senders and receivers are more or less different which is normal if looking at the connection with the perspective of data transmitted.

6.2 Conclusion

To be able to see if a DDoS attack is in preparation or progress we have to correlate all the information that the above scanners provide us with closely. We would need to see what type of connections they are, this is important as we in many DDoS attacks would not expect that the zombies are using their real ip-addresses and hence we would have a lot of half open connections on the machines being attacked filling its memory and finally leading to the impossibility to receive more requests completing the DDoS attack.

I deem the scanners I have constructed to be useful and functional, but in need of being complemented with at least a scanner looking at the connection oriented protocol's flags to see what type of connection is being used as explained in [5], this would greatly increase the reliability of the information my scanners have provided and give the possibility to interpret the results better. The impossibility to do this due to errors in the

implementation of the Netflow data format explained in the last paragraph in chapter 6 has led to this part being left open to further research. I would propose to take the actions proposed in chapter 6.3 to solve this problem which I find important due to that over 94% of the attacks use the TCP protocol in DoS attacks [22].

6.3 Outlook

To give a more complete view of the traffic analysed there is great need of constructing a scanner that shows the number of half open connections per host basis. This could be constructed using another version of the Netflow data format where we have the flags for the connection oriented protocols present.

I would propose plotting the number of retransmitted SYN-ACK packages as well as RST responses and ICMP replies for each ip-address. This would improve the picture the scanners outlined in chapter 5.2 provide.

Other interesting statistics would be to have a graph showing the hosts that try to connect to unused ip addresses. This is often the case when a worm or virus is spreading because they often use randomly generated IP-address-endpoints to try to infect. Finally flow correlation would be of interest to be able to track specific connections and their behaviour.

7 Summary

Multiple scanners have been constructed that can indicate the ongoing or preparation of a DDoS attack signature. The scanners might be useful in other aspects as well e.g. the detection of flashcrowds, virus- and worm-spreading etc. To improve the picture for reliable analysis of DDoS attacks or preparations additional scanners analysing the number of half open connections, flow correlation and connection-attempts to unused ip-addresses have been proposed.

8 References:

[1] CERT Coordination centre, Denial of service attacks.
http://www.cert.org/tech_tips/denial_of_service.html (22 Jun 03)
[2] Schneier Bruce, Secrets and Lies, digital security in a networked world, John Wiley & Sons, Inc 2000, chapter 4 page 43.
[3] William Stallings, Network Security Essentials: Applications and standards, Prentice Hall 2000, chapter 1 page 8.
[4] Gollmann Dieter, Computer Security, John Wiley & Sons, Inc 2000, chapter 13 page 227.
[5] RFC 2827. Example path: http://www.faqs.org/rfcs/rfc2827.html (22 Jun. 03)
[6] Chang Rocky K. C., Defending against Flooding-Based Distributed Denial-of-Service Attacks: A tutorial, The Hong Kong Polytechnic University, IEEE Communications Magazine, October 2002.
[7] Gibson S., Distributed Reflection Denial of Service: Description and Analysis of Potent, Increasingly Prevalent, and Worrisome Internet Attack,
http://grc.com/dos/drdos.htm (22 Jun 03)
[8] A proprietary data standard by CISCO
http://www.cisco.com/warp/public/732/Tech/nmp/netflow/index.shtml (26 Jun 03)
[9] www.switch.ch (02 Jul 03)
[10] Swiss Federal Institute of Technology, Zurich, www.ethz.ch (02 Jul 03)
[11] Plattner B. Wagner A. and Dübendorfer T., DDoSVax In search of a Vaccine for Distributed Denial of Service Attacks, version 1.11.2002, ETH. (TIK/ETH internal document)
[12] Brandford Paul and Plonka David, Characteristics of network traffic flow anomalies, ACM SIGCOMM Internet Measurement Workshop, 2001.
[13] Canonico R., Cotrneo D., Peluso L., Romano S.P. and Ventre G., programming routers to improve network security. OPENSIG workshop, 2001.
[14] Handley M., Kreibich C.and Paxon V., Network intrusion detection: Evasion, traffic normalization and end-to-end protocol semantics, USENIX Security Symposium, 2001.
[15]
http://www.cisco.com/en/US/products/sw/netmgtsw/ps1964/products_implementation_design_guide09186a00800d6a11.html#xtocid7 (24 Jun 03)
[16] http://www.cert.org/advisories/CA-2003-04.html (24 Jun 03)
[17] http://sources.redhat.com/bzip2/ (24 Jun 03)
[18] http://www.tik.ee.ethz.ch/~wagner/ (29 Jun 03)
[19] http://www.ucc.ie/gnuplot/gnuplot.html (24 Jun. 03)
[20] http://www.gnu.org/software/bash/bash.html (24 Jun. 03)
[21] http://www.tik.ee.ethz.ch/~ddosvax/ (26 Jun 03)
[22] D.Moore, G. Voelker and S. Savage “Inferring Internet Denial-of-Service Activity” Proc. 10th USENIX Sec. Symp. 2001