ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Daniel Reichle
<daniel@reichle.li>

Analysis and Detection of DDoS Attacks in the Internet Backbone using Netflow Logs

Diploma Thesis DA-2005.06
June 2005 to October 2005

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

Abstract

The ability to detect DDoS attacks in the Internet backbone in real-time is crucial for network operators in order to provide an infrastructure of high quality.

In this thesis, an algorithm for the near real-time detection of massive flooding attacks as well as TCP SYN flooding attacks has been developed and implemented as an UPFrame plug-in.

The algorithm has been validated successfully by replaying NetFlow v5 logs of the SWITCH backbone border routers, containing known ICMP, UDP and TCP SYN flooding attacks from the past.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

Denial of Service (DoS) attacks are one the major annoyance of today's Internet. Their sole purpose is, as the name indicates, to prevent an Internet server from doing its work- which is to provide a service to other Internet hosts. The impact of DoS attacks can vary from annoying regular users of the affected server, for example a web site, to financial and image damage of the server's operator.

Whatever reasons motivate the launch of DoS attacks, they seem to be quite popular these days [17] and the Internet technology which was not designed with security concerns provides little means to prevent them. The following section gives a quick overview of different kinds of DoS attack.

1.2 DoS attacks

There exist many different kinds of DoS attacks [24], [22], which can be divided into two categories: Attacks that exploit a network protocol to prevent the attack victim from processing other requests, and attacks that are simply consuming all the available bandwidth of the victim. For the types of flooding attacks that are dealt with in this thesis, namely UDP flood, ICMP flood and TCP SYN flood, a summary is given in the following.

1.2.1 UDP flood [35]

An attacker sends a large number of UDP packets to one or several ports of the victim, which eventually renders the attacked host unable to process the incoming packets.

1.2.2 ICMP flood

An attacker sends a large number of ICMP echo requests to the victim to consume all its available bandwidth.

1.2.3 TCP SYN flood [36]

As opposed to the two attacks described above, the TCP SYN flooding attack is a protocol exploiting attack. It exploits the three-way handshake of the TCP protocol by sending many TCP SYN requests to a victim. Because the attacked host maintains half open ports to wait for the connections to be established, it eventually runs out of resources.

1.2.4 Distributed attacks

Through the use of automated tools it is possible to lead all the attacks discussed above in a distributed way. This is done by compromising poorly secured hosts, so called zombies, and install clients of the attack tool. Then the attacker can call all the zombies to launch an attack on the same host simultaneously. See [8] for an overview of different techniques used in distributed DoS attacks.

2 Problem

2.1 Problem statement

DDoS attacks are a threat to the Internet. They decrease the service quality of Internet services– therefore it is important to have means to prevent or at least mitigate them. To apply countermeasures against Denial of Service attacks, or even to only analyze them, the first crucial step is to detect such attacks.
In the DDoSVax project [38], the environment to detect DDoS attacks is the collection of flow-level network data, captured by the SWITCH [39] backbone border routers. The data is exported in Cisco NetFlow v5 format [27] to UPFrame [37], a framework for real-time processing of NetFlow data.

The task given for this thesis was to develop an algorithm which detects (Distributed) Denial of Service attacks on the basis of analyzing NetFlow records. The algorithm was to be implemented as an UPFrame plug-in for real-time processing, which restricted the possibilities in terms of resource usage and processing time.

2.2 NetFlow

In NetFlow data, no single IP packets are visible. Network traffic is aggregated to flows instead, which collect consecutive packets with the same source and destination IP address, same protocol and ports. A flow is completed, if no data is seen during the inactive timeout, if the flow exceeds the active timeout (15 minutes for our data), or after a FIN or RST packet in TCP connections. Besides the source and destination IP address, the used protocol and source and destination port, the information contained in each flow record includes start and end time of the flow, the number of packets and the total number of bytes in the IP layer. The used TCP flags, however, are not visible in our data.
The drastic reduction of data offered by NetFlow, compared to single packet processing, leaves much more space for experimenting with very large amounts of network traffic data.

3 Related work

3.1 Overview

Detection of Denial-of-Service attacks is a complex problem which has been approached in various ways. An overview of the problem can be found in [8]. While detection of DDoS attacks on the victim's network is easy, it provides very little means to prevent them [16]. Detecting DDoS attacks in the Internet backbone is much more difficult and is best done employing a distributed mechanism [8].
Of the different approaches of DDoS defense mechanisms [24] we will focus here on the reactive mechanism based on detection of traffic anomaly. In this chapter, different (D)DoS defense mechanisms are presented and compared. "Defense mechanism" is a very general term and had been chosen to include detection and defense algorithms- it can be an algorithm applied to a stream of network packets, as well as a framework of autonomous nodes spread over the Internet backbone in which each deploys such an algorithm. In the latter case, the attack mechanism lies in the concept of a distributed detection.
The presented mechanisms differ in many ways, such as operation on single packets or on aggregated flows, deployment at the attack's victim or close to the attacker, or the ability to adapt thresholds to the current state of the system's environment, summarized in Table 3.1. A short description of each of the mechanisms is given in Sections 3.2.1 to 3.2.11.

Method	Action			Location				Target		Input					Signature			Update		Remarks	
	Detection	Prevention	Defense	Victim's network	Internet backbone	Near attack source	Distributed	Known attacks	Unknown attacks	Payload	Header	Single packet	Netflow log	Exact pattern matching	Threshold	Congestion	adaptive	non-adaptive	Implementation [a]	
Backscatter [26]	X				X			X			X				X			X	X	[b]
Kolmogorov complexity [21]	X			X					X	X		X			X			X	T	
DDoS on ISP networks [1]	X				X		X		X		X	X			X		X		T	
Detecting SYN Flooding [32]	X							X			X	X			X		X		T	
Detecting SYN Flooding [34]	X				X			X			X	X			X		X		T	
Adaptive change-point detection [5]	X								X		X	X			X		X		T	
DDoS detection using MIB [6]	X			X		X	X	X							X			X	T	[c]
CATS [11]	X				X		X	X	X			X		X	X		X		-	[d]
Detecting Pulsing DoS attacks [23]	X			X					X		X	X			X		X		T	
Space-time network flow patterns [2]	X				X		X		X		X	X			X		X		T	
Sharing beliefs [30]	X				X		X	X			X	X			X		X		T	
Statistical approaches to DDoS attack detection [13]	X				X				X		X	X	X		X		X		X	
Hop-count filtering[20]	X		X					X			X	X		X				X	T	
COSSACK [28]	X		X	X		X	X		X		X		X		X		X		T	
D-WARD [25]	X		X			X			X		X	X	X	X	X		X		T	[e]
Pushback [19]	X		X	X			X		X			X				X		X	T	
Ingress filtering [14]		X	X		X			X			X	X		X				X	X	
Traffic control system using traffic ownership [12]		X	X		X		X	X	X		X	X		X				X	X	
Aguri [10]			X		X				X		X	X			X		X		X	
Packet funneling [7]			X	X					X		X	X				X		X	X	

[a] T: Implemented in simulation environment, X: Tested with real traffic data
[b] Doesn't work for reflector attacks.
[c] System doesn't collect packets but collects MIB (Management Information Base) data from the surveilled routers.
[d] Not a new algorithm, but a combination of existing ones.
[e] Deployed at source end networks.

Figure 1: Overview of existing (D)DoS defense mechanisms

3.2 Algorithms for DoS attack detection and defense

3.2.1 Backscatter [26]

Backscatter analysis takes advantage of the fact that for direct DoS attacks, attackers commonly spoof the source IP address at random. Attacks can be detected by monitoring a large enough IP address range and detect unsolicited responses from a victim.
The algorithm assumes that the source address is spoofed at random, that attack traffic is delivered reliably to the victim, that backscatter is delivered reliably to the monitor and that unsolicited packets observed by the monitor represent backscatter.
The algorithm is not able to detect (D)DoS reflector attacks.

3.2.2 Kolmogorov Complexity [21]

The algorithm uses Kolmogorov Complexity to correlate traffic flows in the network and detect possible DoS attacks. The underlying assumption is that information, comprising observations of actions with a single root cause, whether they are faults or attacks, is highly correlated and therefore has a high compression ratio.

3.2.3 DDoS detection on ISP networks [1]

The proposed detection mechanism is distributed: each router detects traffic anomalies using profiles of normal traffic constructed using stream sampling algorithms. Normal traffic profiles are created by sampling traffic over a relatively long time, while current traffic profiles are constructed by using smaller time windows. Whenever the current profile does not corroborate with the normal one, a router becomes suspicious. Routers aggregate the suspicions received from all other routers before deciding whether a certain traffic aggregate belongs to an attack or not.

3.2.4 Detecting SYN Flooding (Siris, Papagalou) [32]

Siri and Papagalou present two anomaly detection algorithms for detecting TCP SYN attacks: an adaptive threshold algorithm and a particular application of the cumulative sum algorithm for change point detection. The adaptive threshold algorithm compares the number of SYN packets received over a given interval to the estimated number based on recent measurements. To raise an alarm, the threshold has to be exceeded consecutively.
The CUSUM algorithm uses the difference between the number of SYN packets in a time interval and the estimated number for the same interval as a Gaussian random variable. A SYN flooding attack is then detected using the cumulative sum based on the likelihood of the momentary value being caused by a change in the mean traffic rate.

3.2.5 Detecting SYN Flooding (Wang, Zhang, Shin) [34]

The SYN flooding attack detection mechanism proposed by Wang, Zhang and Shin is based on the protocol behaviour of TCP SYN-FIN (RST) pairs at leaf routers that connect end hosts to the Internet. The difference between the number of observed SYN and FIN (RST) packets is observed and a CUSUM algorithm is applied for a change point detection. To reduce the impact of different access patterns of different sites, the difference between the number of SYNs and FINs (RSTs) is normalized by an estimated average number of FINs (RSTs).

3.2.6 Adaptive change point detection [5]

Adaptive anomaly detection system based on the detection of change-points in the observed traffic pattern. The proposed detection methods employ statistical analysis of data from multiple layers of the network protocol for detection of changes in the statistical models of traffic. The information regarded includes the number of TCP packets categorized by size or type, the numbers of UDP packets and their sizes, the source and destination port for each packet, etc.

3.2.7 DDoS detection using MIB [6]

The proposed system detects DDoS using information from MIB (Management Information Base) traffic variables for attacker and target. Signatures to match for attack detection were determined from known attacks. On the attacker's side, a DDoS attack should be detected prior to its launch by identifying MIB-based precursors.

3.2.8 CATS [11]

The CATS (Cooperative autonomous attack detection) system comprises individual detection systems consisting of a network monitoring part and an attack detection part. Each detection system captures network data and applies both statistical anomaly detection and signature based mechanisms to detect DoS attacks. The anomaly detection engine compares long-time behavior to short-time behavior of statistical parameters not further specified. The signature matching engine is based on tools like Snort and Bro. An enhancement of the detection quality is achieved through coupling multiple autonomous systems, which share their information.

3.2.9 Detecting Pulsing DoS attacks [23]

Pulsing DoS (PDoS) attacks are detected at the victim's network using two anomalies caused by PDoS attacks, namely the fluctuation of the incoming data traffic, and the decline of outgoing TCP ACK traffic. To monitor these parameters, the input is sampled using a discrete wavelet transform (DWT). A CUSUM algorithm is then employed to detect abrupt changes in the statistics.

3.2.10 Space-time network patterns [2]

Method to detect DDoS attacks in the Internet backbone by a set of nodes. The idea is that for a DDoS attack, a direction of the attack is observable, since the data tends to aggregate from the distributed sources towards the target. For monitoring nodes, the number of packets sent by a node are tracked using a vector with one element for each neighbor. To detect a change in the direction of the node's flow, a generalized likelihood ratio algorithm is applied. Statistics of different nodes are then correlated to decrease the detection delay given a fixed false alarm rate probability.

3.2.11 Sharing beliefs [30]

Approach to detect distributed reflector attacks. Potential reflectors broadcast a warning message to other potential reflectors if abnormal traffic is observed. The detection decision is then made based on the information from multiple potential reflectors. A reflector detects an abnormal network behaviour using a CUSUM algorithm on the number of RST packets that have a SYN/ACK state in the outgoing connection. To detect other attacks than TCP, ICMP port unreachable packets are considered. The agents cooperate by sharing their beliefs about potentially suspicious traffic, leading to a general decision.

3.2.12 Statistical approaches to DDoS attack detection [13]

Algorithms are proposed that detect DDoS attacks on the basis of statistical properties of specific fields in the packet headers measured at various points in the Internet backbone. To detect changes in the statistics of a parameter, its entropy is calculated consecutively for a sliding window of packets. For the comparison of distributions where the number of possible values is small, Pearson's chi-square test can be applied as well. Examples of parameters are source and destination IP address, TCP/UDP ports, datagram length and TCP window size.

3.2.13 Hop-count filtering [20]

Hop-count filtering is a method to filter spoofed IP packets by examining the Time-to-Live (TTL) value in the IP header. Since most operating systems use only a few selected initial TTL values,

the initial TTL value can be guessed from the observed TTL value, and therefore the hop-count of an IP packet can be obtained. Using a table that maps IP addresses and hop-count values, it can be checked if the packet's hop-count value matches the hop-count of the source address-in case of a mismatch the source address is likely spoofed.

3.2.14 COSSACK [28]

COSSACK is a distributed DDoS detection and response system. An instance of its watchdog program monitors a network and detects attacks using any intrusion detection mechanism. When a watchdog detects an ongoing attack, it broadcasts information about it to other watchdogs indicating the attacking source networks. Each source network watchdog then tries to find out if there exist zombies within its infrastructure and deploy countermeasures if necessary.

3.2.15 D-WARD [25]

D-WARD is a DDoS defense system that is deployed at the source-end network to prevent the machines from participating in DDoS attacks. Network flows not complying to predefined models are rate-limited dynamically. The monitored data is aggregated traffic from and to each local host for each of its active connections. Considered statistics are TCP packets sent/received ratio, ICMP echo, time stamp, and information request and reply packets sent/received ratio and number of UDP "connections", packets and sending rate per destination.

3.2.16 Pushback [19]

Pushback is a mechanism employing aggregate-based congestion control. It is implemented on routers to prevent bad traffic from congesting links in a recursive way. The idea is that traffic belonging to an attack is most easily recognized near the victim, and therefore the router closest to the victim can identify the links which deliver the traffic causing the congestion. It then drops traffic from these links and tells the routers forwarding the bad traffic to rate-limit traffic. These routers propagate the request along the attack path towards the attack source(s) requiring that each router on the attack path supports pushback.

3.2.17 Ingress filtering [14]

Ingress filtering applied by routers, is a method to mitigate attacks involving IP address spoofing. The concept is that routers connecting hosts to Internet service providers check if the source address of every packet to be forwarded is from within the network it is coming from. Packets with source addresses from outside the source's prefix range are dropped. With ingress filtering applied, it is still possible to use spoofed source addresses for an attack, but since the spoofed addresses have to be from within the attacker's prefix range it is easier to track the attacker down, or at least the victim can block a range of source addresses until the problem is solved.

3.2.18 Traffic control system using traffic ownership [12]

The authors propose a distributed Internet traffic control system that controls network traffic close to routers and that is deployed by various network operators. The used fundamental concept of traffic ownership ([12]) guarantees that deployed traffic control and filter rules have no negative impact on the network and that collateral damage is prevented. For the case of DDoS, a server operator preventively or reactively can inject rules into the traffic control system to block others from using his registered IP addresses as source addresses, which immediately blocks source spoofed packets that are required for e.g. a DDoS reflector attack. Route-based filtering, rate-limiting and other mechanism can be deployed to mitigate more traditional DDoS attacks. Due to the distributed nature of the system, malicious traffic can be stopped close to the attack source.

3.2.19 Aguri [10]

Aguri is a software package that uses a traffic profiling technique in which records are maintained in an address prefix-based tree. To generate an entry, flows are aggregated until the aggregation has a certain volume, then four summary profiles are produced for source addresses, destination addresses, source protocols and destination protocols. A summary output is produced by traversing the tree and aggregating entries. Hostile activities are identified in the aggregated profiles.

3.2.20 Packet funneling [7]

Packet funneling is a load balancing approach to mitigate the impact of DDoS attacks at the victim's side by privileging regular incoming traffic. The concept's underlying assumption is that DDoS attacks use IP spoofing resulting in a stream of packets with different, non-recurring IP source addresses, opposed to IP addresses recurring at small intervals in legitimate traffic. To distinguish regular traffic from attack traffic, packet funneling keeps track of the arrival times of incoming packets using an Active IP (AIP) table. If the number of active IP addresses exceeds the size of the table, new addresses are added to the Waiting Matrix. While arriving packets of addresses in the AIP are forwarded, packets of addresses in the Waiting Matrix are delayed. When the timeout value set on every packet's arrival expires for an address in the AIP, the entry gets dropped, and one of the address in the Waiting Matrix will become active instead.

4 Algorithm

4.1 Basic idea

Distributed Denial of Service flooding attacks concentrate network traffic from many hosts, each not needing to provide a high bandwidth network link, on a single victim host. Hence the closer to the victim a DoS detection algorithm is deployed, the easier it is to detect an attack. The idea of the algorithm developed in this thesis is to take advantage of this fact when finding a way to detect DoS attacks in the provided Internet backbone traffic.
Algorithms for detection of DoS attacks in the backbone developed so far are mostly based on statistical anomalies of IP packet header fields(see Figure 3.1). While some statistical analyses can be applied to network traffic at flow level as well (e.g. flow length, average packet size for a certain protocol), the approach chosen here also takes advantage of having a global view of a whole autonomous system (AS) and aims at detecting DoS attacks by observing potential victims. To detect an attack, the ratio of incoming to outgoing packets of a host is considered as a very general way to describe a Denial of Service attack. The idea is that while for regular network connections traffic is always bidirectional to a certain degree, in case of a DoS attack traffic is highly asymmetric in terms of the packet ratio. Thus, the primary objective of the algorithm presented here is to find hosts with a exceedingly high in/out packet ratio.

4.2 Description

4.2.1 Aggregating and storing the input data

First of all, the continuous input stream of flows has to be aggregated for discrete time intervals. Here it has to be considered that the flows exported in the NetFlow logs are not strictly ordered in chronological order according to their end time. Furthermore, flows can be spread over several intervals, so there has to be taken care of assigning all the packets of a flow to the respective intervals. Since only the start and end time of a flow is known but not the time distribution of the contained packets, a linear distribution is assumed as an approximation. The possibility of flows exceeding one interval makes it necessary to store data for several intervals, so no further measures are taken on the basis of data that might not be appropriate due to possible subsequent flows. It is also necessary to store the number of packets for both sender and receiver of a flow since the sender could be falsely considered a victim after receiving many packets if the number of packets sent before was discarded.
Besides the number of incoming and outgoing packets, the algorithm keeps track of the number of sent and received bytes and the number of incoming and outgoing short TCP flows for every sender and receiver IP address of input flows. The granularity of the observed addresses can be set to single host addresses or address ranges through specifying a subnet mask in the configuration.

4.2.2 Observation time intervals

The length of the flows in the input is dependent on the active timeout of the NetFlow exporting router, in our case 15 minutes. That means the algorithm has to keep data for many intervals, depending on the interval length if we want to distribute also traffic reported in long flows to our e.g. 1 minute interval statistics. However, considering that most of the flows are shorter than 2 minutes (see section 5.3), it is not necessary to store that much data. Instead, a two-phase method is applied.
In the first place data is only stored for a small number of intervals, and only hosts whose packet ratio exceeds a threshold after that short observation period are considered interesting and therefore are to be observed for the maximum flow length to await possible subsequent flows containing packets for the respective interval.
For long flows to an IP address which is only observed for the short period of time, only the last intervals are stored. This proceeding delays the detection of an attack for up to the difference between the short period and the maximum flow length. A packet ratio check of all stored IP addresses is done after each interval, and the observation time of high ratio hosts is extended. If the high ratio of a host is lowered under the threshold after processing a flow sent by it, the host is no longer considered interesting and therefore put back to short time observation.

4.2.3 Attack detection

If a host has still a high ratio value for an interval after waiting for the maximum flow length, it becomes an attack candidate. Before it is logged as an attack victim, however, it has to pass some more tests. To be considered a victim of a flooding attack, its packet ratio has to exceed the attack threshold; furthermore the numbers of received packets and bytes have to exceed the respective thresholds. Involving more parameters makes it possible to suppress false alarms in a flexible way.
To detect victims of TCP SYN attacks, the algorithm employs a second test, since in this case an attack is not necessarily determined by the victim receiving much data. For this case, the ratio of the number of incoming to outgoing short TCP flows of a host during the interval is considered. If this ratio exceeds the respective threshold as well as the number of incoming short TCP flows, the host is logged as a TCP victim.
See Section 5.3 for a discussion about the parameters of the algorithm.

4.2.4 Analysis of the algorithm: strengths and weaknesses

Strengths

- The presented algorithm provides an effective method to detect Denial of Service attacks in the vast amount of input data.
- Its low memory and CPU usage suits its purpose as a realtime plug-in for UPFrame.
- It is highly scalable in terms of memory usage and speed.

Weaknesses

- Its main weakness lies in its inability to see the processed data in a context. Even though with the in/out packet ratio a parameter independent from individual system specifications is used, the algorithm still relies on static thresholds, not considering, for example, the bandwidth of hosts.
- The quality advantage through detecting changes in the observed values has been dropped in favor of performance.
- The algorithm depends on having a full view on the traffic between an observed host and its potential attackers, which is given for the traffic provided by all SWITCH border gateway routers. Running the algorithm with input from only a subgroup of these routers, however, would make it impossible to draw appropriate conclusions; due to asymmetric routing, traffic from and to a host can have different routes.
- Another weakness is the possible delay in attack detection caused by the initial observing of only a short period of time. This delay adds to the delay of the data, which is inherent to the exported NetFlow data.

See section 8 for a discussion about possible improvements.

5 Validation on known attacks

The idea described in the previous chapter first had to be validated. To see if the considered parameters show the expected behavior in case of an attack, analysis was done on Netflow data from the archive, captured during later discovered attacks. Since for these attacks the victim was known, it was possible to filter the network traffic data so traffic from and to the attack victim could be observed. For this purpose, the host analyzer tool was developed.

5.1 The host analyzer tool

The host analyzer tool reads NetFlow logfiles and observes network traffic from and to a specific IP address. After each interval it outputs the number of packets and bytes from and to the host, along with the average number of bytes per incoming packets and the number of active incoming flows during the interval. Before quitting it also outputs flow length statistics as well as protocol statistics.

5.2 Analyses of known attacks

In the following, the results of the analyses of known attacks are presented. For each different attack, the graph of the ratio of incoming to outgoing packets is shown along with the number of incoming bytes. If no incoming packets were seen, the number was set to 1 for the calculation of the ratio to avoid a division by 0.

5.2.1 UDP to IRC server

Before and after the attack the IRC server received between 120’000 and 220’000 packets per minute at a in/out packet ratio between 0.98 and 1.36. During the attack that started at 23:26 the packet ratio reached at peak value of 137.58 and 1058 MB of incoming data in one minute (see Figure 16). The attack was led by a single host sending UDP packets at high rate.

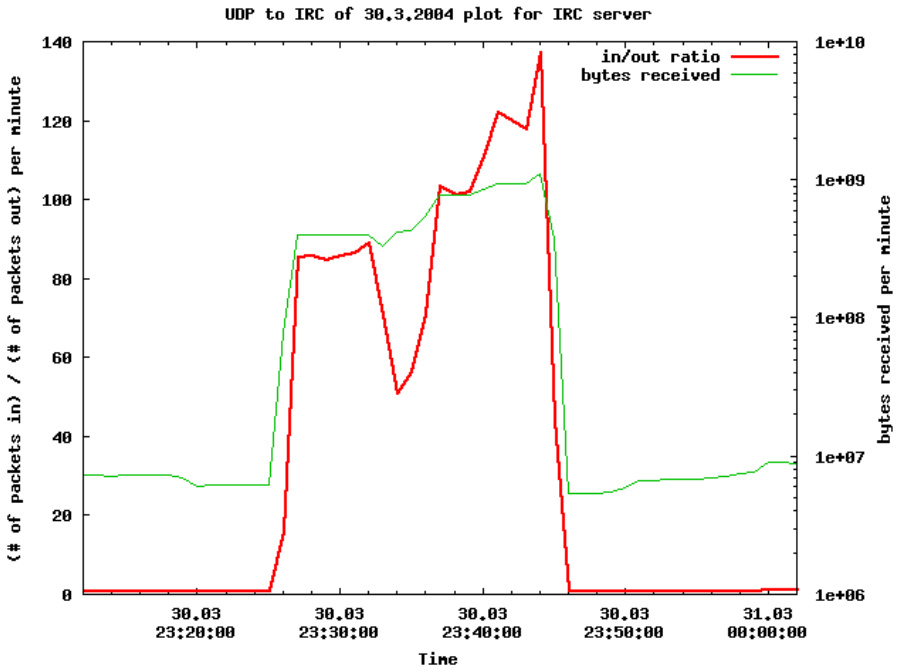

Figure 2: UDP to IRC

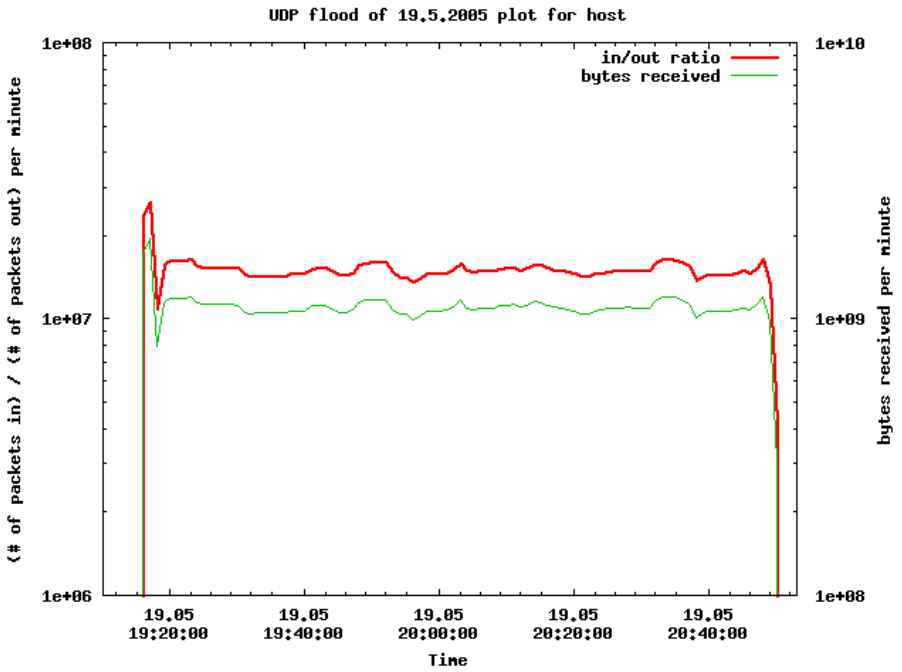

Figure 3: UDP flooding 1

5.2.2 UDP flooding 1

The victim of this UDP flooding attack did not send a single packet during the 3 hours of the analysis. The attacker was sending UDP packets of constant size (73 bytes) to different ports of the victim. Since the victim was not responding, the in/out packet ratio equals the number of incoming packets here (see Figure 3).

5.2.3 UDP flooding 2

In this UDP flooding attack (see Figure 4), the attacker sent more than 5'000 MB of data to several ports at the victim within 35 minutes (including intermissions). The packets did not contain any payload. The in/out packet ratio was almost constantly around 100'000, with the victim sending about 50 packets per minute while receiving 5 millions.

5.2.4 UDP to web server

The webserver received more than 11'000 MB in 50 minutes during this UDP flooding attack. Before and after the attack, the webserver had a constant in/out packet ratio of 1.0 with receiving around 15 packets per minute before, and around 4 packets after the attack. The number of packets received per minute reached a peak value of 1 million during the attack. The zigzag shape of the in/out ratio curve (see Figure 5) is caused by the fluctuation of the number of packets sent by the victim (between 0 and 9 per minute). The received packets did not contain any payload.

5.2.5 UDP to web proxy

In this attack, the attacker sent more than 5'000 MB to a UDP port of the victim during 19 minutes. As in the two previously described attacks, the UDP packets did not contain any payload. The victim did not send any packets during most intervals, but sent more than 5'000 packets in two intervals of the attack, resulting in a abrupt fall of the in/out packet ratio (see Figure 6).

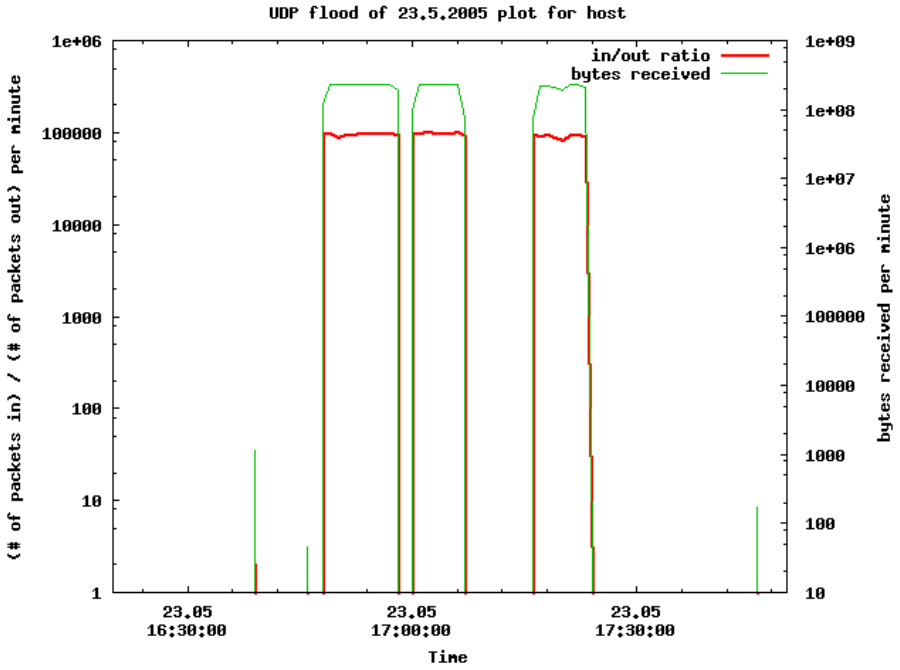

Figure 4: UDP flooding 2

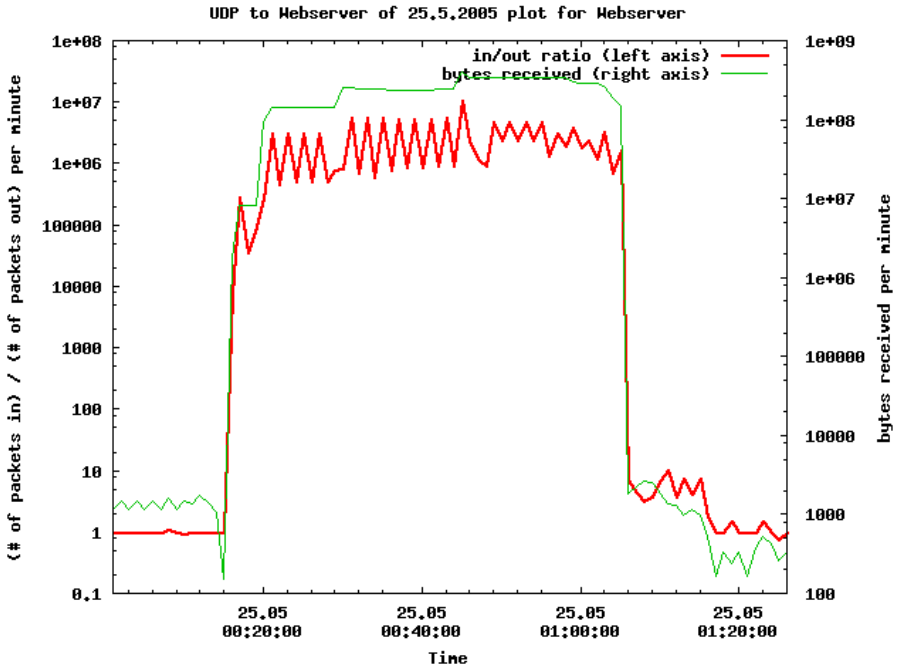

Figure 5: UDP to web server

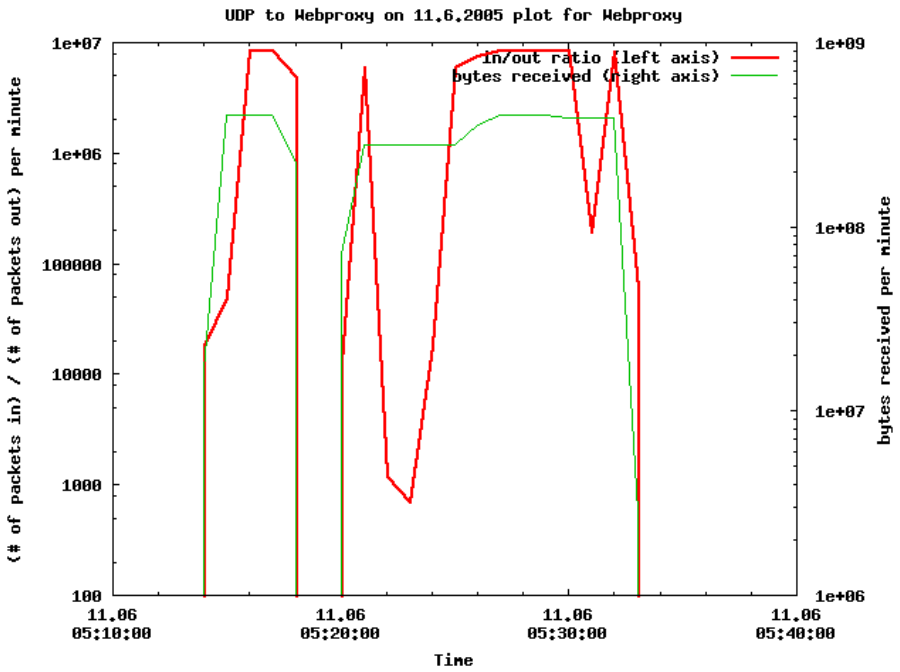

Figure 6: UDP to web proxy

5.2.6 ICMP flooding

The only example of an ICMP flooding attack shows how an attacker sent ICMP packets with payload (packet size on IP layer of 1’500 bytes) to the victim during 1 hour and 45 minutes with a nearly constant rate of around 68’000 packets per minute (see Figure 7). In total, 10’211 MB were transmitted. The victim did not send any packets during the time of this analysis.

5.2.7 TCP to myeth

The attack on the myeth server is the only known TCP attack that was available for analyzing. In contrast to the UDP and ICMP attacks described previously, this attack was conducted by sending many TCP SYN requests instead of a huge amount of data. The incoming traffic during the attack, however, was highly asymmetric and lead to an in/out packet ratio of 50, compared to a ratio of around 1 before and after the attack (see Figure 8). To see if the effect of high in/out packet ratios were still observable when counting packets for a subnet instead of a single host, the analysis of the attack has also been done with subnet masks 255.255.255.240 (figure 9), and 255.255.255.0 (figure 10). The two figures show that attacks can be quite apparent when watching the in/out packet ratio for a single host, but involve so few packets that they disappear in the statistics as soon as other hosts are added to the observed IP address range.

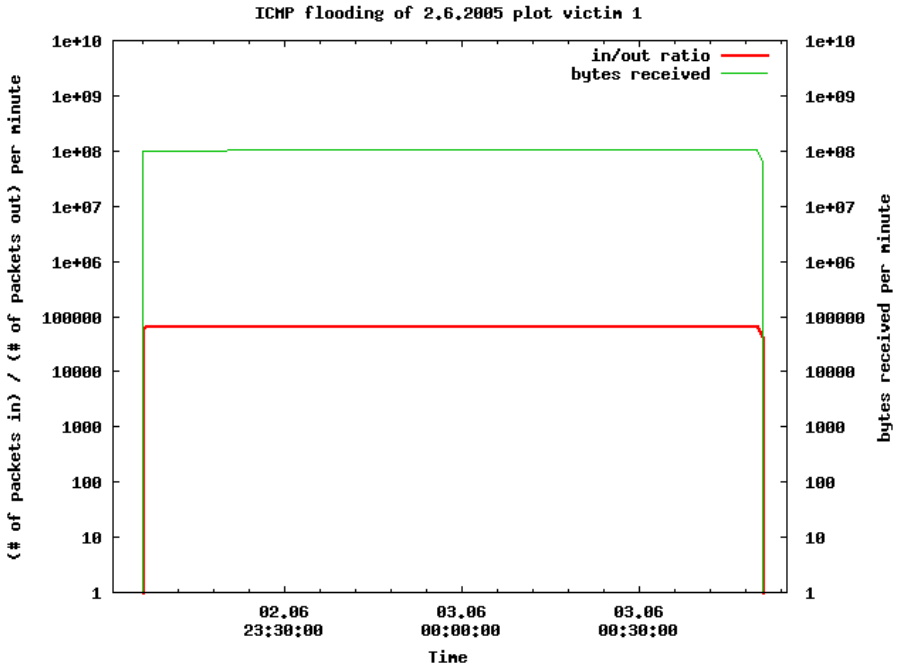

Figure 7: ICMP flooding

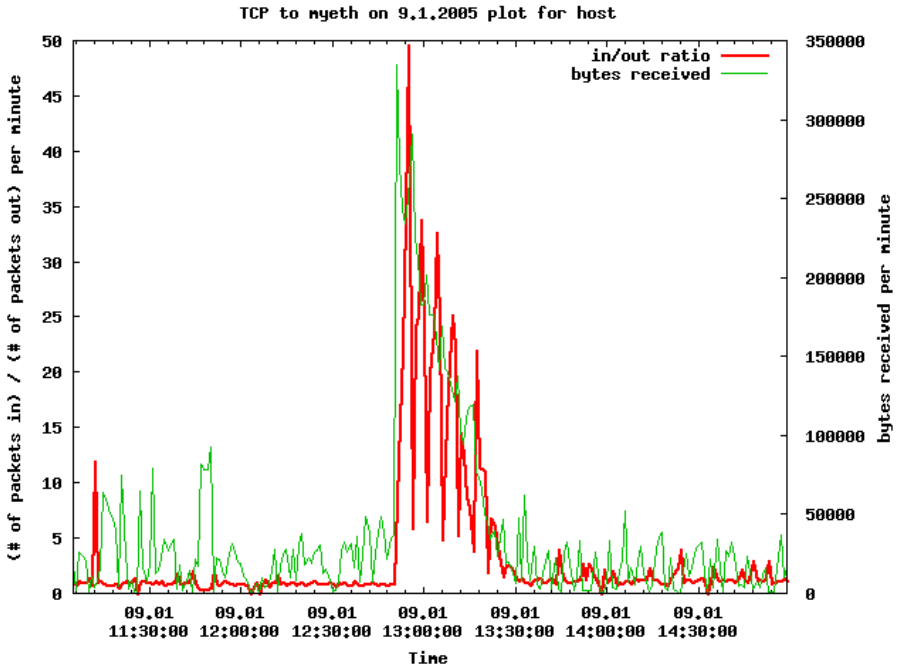

Figure 8: TCP to myeth

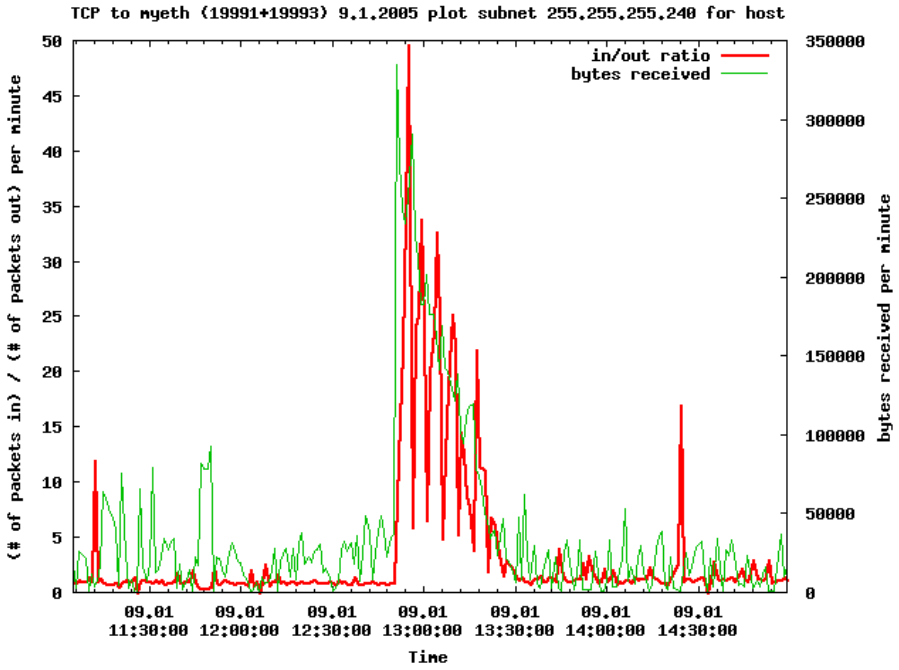

Figure 9: TCP to myeth (subnet 255.255.255.240)

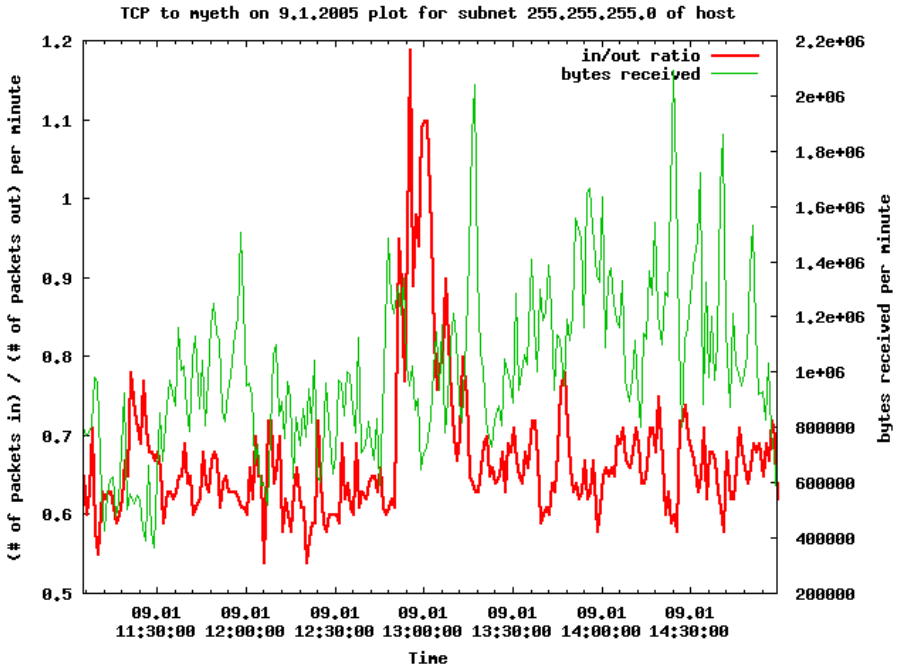

Figure 10: TCP to myeth (subnet 255.255.255.0)

5.3 Characterization of attacks

The analyses of the known attacks clearly showed that the in/out packet ratio for an Internet host serves the purpose of indicating the ongoing of a Denial of Service attack. With the clarification of this issue, however, the question of the parameters that distinguish attacks from regular traffic still remained.

The assumption that network connections are always "bidirectional to a certain degree" (Section 4.1) has now to be specified in numbers. Observation of the www.ethz.ch web server on a random day shows that for http traffic the average of the in/out packet ratio is 0.9 at an average of 876 incoming packets per minute (see Figure 11).

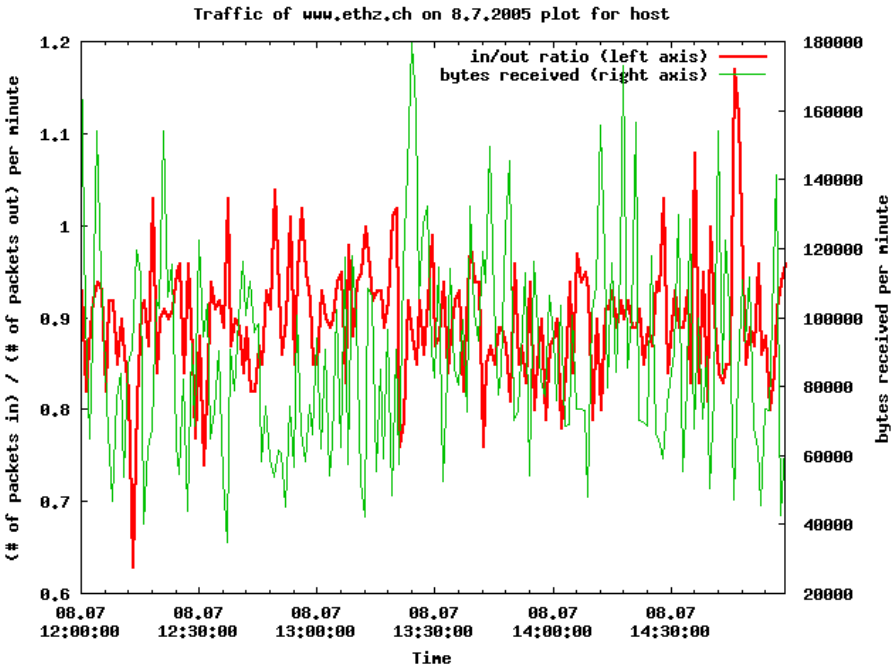

Figure 11: www.ethz.ch on a normal day

For connectionless protocols like UDP, however, we can not expect the ratio to be this balanced. To decide when a packet ratio is unusually high, empirical tests have been done on the NetFlow log files containing the known attack.

For testing the detection algorithm, the in/out packet thresholds for further observation of a host, as well as the threshold for the decision whether a host is under attack or not, were set arbitrarily based on these empirical statistics. The decision to use two thresholds allows the appliance of a low threshold to not abandon surveillance of possible victims early without giving away the chance of suppressing false alarms. Regarding the data of the known attacks described in this chapter, the threshold for the observation in/out packet ratio has been set to 15, and the threshold for an attack alarm to 100. For the detection of TCP SYN attacks, the threshold for the in/out ratio of short TCP flows has been set to 15.

Furthermore, including the number of received packets and bytes helps eliminating false positives. The thresholds chosen here were 30’000 for the minimum number of packets, 50 MB for the number of received bytes and 1’000 for the minimum number of incoming short TCP flows. For running the detector plug-in, these thresholds can be set in the configuration file.

6 Implementation

6.1 Overview

The algorithm described in Chapter 3.2.20 has been implemented in a program called DDoS-Detector. The program comes in two versions- a stand alone version that reads NetFlow data from files or from a pipe, and a plug-in version that processes data from UPFrame. The changes in the plug-in version are described in Section 6.3, the features described in section 6.2 apply to both versions. See Section B for a description of the program parameters mentioned here.
The software is written in C, and comprises about 2'000 lines of source code. Its logical structure of the software is depicted in Figure 12. The operation of the program is simple. Its main control

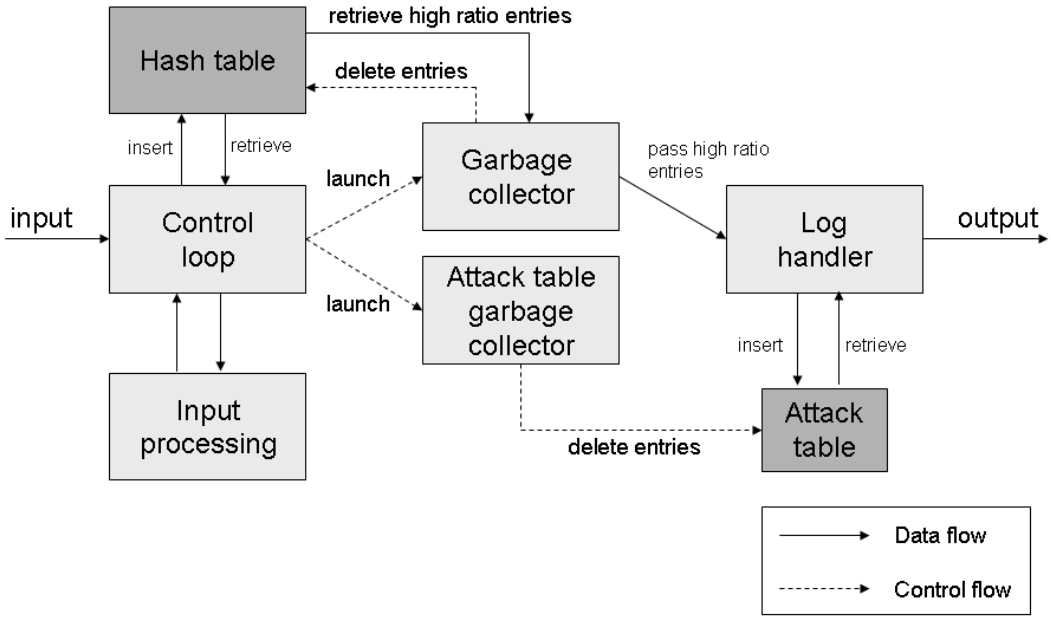

Figure 12: Logical structure of the software

loop is executed for every NetFlow v5 record in the input data. Here, the hash table is accessed to retrieve the entries for the source and destination IP address of the currently processed flow which then are handed to the input processing part, along with the new flow. After the two entries are updated, the main control loop inserts them back into the hash table.
The garbage collection and attack table garbage collection are called regularly and delete obsolete entries in the hash tables if necessary. At the end of each interval, it is called to also check the packet ratio of all entries when iterating the hash table.

6.2 Implementation details

In the following subsections, a description of the used data structures and the main logical parts of the software is given.

6.2.1 Data structures

The central data structure is the hash table that constantly stores all the relevant traffic data from the processed flows. The hash table contains an entry for every IP address that is currently ob-

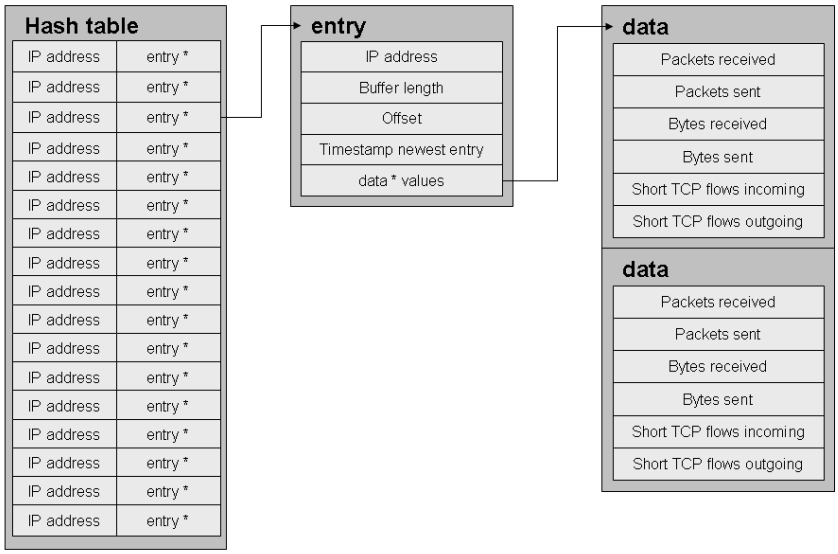

Figure 13: Data structures

served. These entries manage a ring buffer of "data" structs that store the actual data for every time interval the respective IP address was observed: sent/received packets, sent/received number of bytes, number of outgoing/incoming short TCP flows. The interrelation of these structures is shown in Figure 13.

6.2.2 Control loop

The control loop is the central part of the program. In the stand alone version, the control loop is in the "main" method. It iterates the list of given input files, and processes every NetFlow flow contained in the input. For each flow, it checks if entries for sender and receiver IP addresses already exist in the hash table. If so, the entries are retrieved, otherwise new ones are created. The two entries are then passed to the input processing functions, along with the newly received flow. The control loop also keeps track of the time and calls the garbage collector after every time interval.

6.2.3 Input processing

The task of the input processing part is to update the entries for the source and destination IP address of the new flow by adding the flow's information appropriately. This involves splitting the data of the new flow into time intervals according to the proportions of the length of the flow in the different time intervals, if the new flow does not start and end in the same time interval.
The updated entries are then returned to the control loop.

6.2.4 Garbage collection

The garbage collector iterates the hash table and checks for each entry if the latest update is older than the entry's observation time. In that case, the entry is obsolete and can be deleted.
Since the garbage collector has to iterate the whole hash table anyway, it also takes over the task of checking the in/out packet ratios of all entries. The relevant data entry for that is the one whose time stamp's age matches the length of the entry's observation time. If the packet ratio of a short time observation entry exceeds the threshold, the entry is enlarged to be observed for the upcoming intervals as well. In case of a long time observation entry, the host is considered an attack candidate and the entry is handed to the log handler.

6.2.5 Log handler

The log handler checks if the entry also matches the other attack criteria (in/out packet ratio above attack and number of incoming bytes above attack threshold, or in/out short TCP flow ratio and number of incoming short TCP flows above attack threshold), and if positive, the attack table is queried to check if the host is already under attack. If there exists already an entry in the attack table, the entry is updated, otherwise a new entry is created. The relevant information about the current attack is then written to the standard out.

6.3 Realtime issues

To meet the requirements of an UPFrame plug-in, a few modifications had to be done. The problems evolving with realtime processing were the need of a memory limitation, the handling of possible data loss and the synchronization of the program with its input data. The realtime issues addressed for the implementation of the algorithm in an UPFrame plug-in are described in the following subsections.

6.3.1 Data loss

Data loss is not a problem that can be done anything about, but since it can result in erroneous conclusions, it is crucial for the detector program to know when the input data is possibly incomplete. The plug-in version of the program therefore maintains an array representing the intervals for the maximum observation time, where in case of a possible data loss a flag is set for the respective interval. This array is consulted before every log entry, so the possibility of data loss at the time of the logged event appears in the log as well.

6.3.2 Synchronization

The plug-in takes its awareness of time from the "last packet" time stamps of the input flows. That means the control loop processes input flows until a flow has a "last packet" time stamp that is not within the current time interval. The time elapsed on the local clock since the last time interval change is then compared to the time interval length to set the program's processing speed state. After executing the garbage collection, the start of the current interval is shifted forward by the time interval length.
To check if the input is synchronous, the program counts the flows whose "last packet" time is not within the current time interval. If the percentage of such asynchronous flows is above 40 %, a synchronization problem in the input is assumed, and the possibility of data loss is noted.

6.3.3 Advanced garbage collection

In order to be able to guarantee an upper bound of memory usage, the garbage collector was enhanced. The idea is that the more the program's memory usage approaches the given upper bound, the more restrictively the garbage collection deletes entries from the hash table. State variables have been introduced to keep track of the current state of the memory usage and processing speed relative to the input rate. The memory usage is checked periodically by the control loop and and the current memory state variable is set accordingly, before the garbage collection is called.
The garbage collector first checks the current memory and processing speed state and decides how to proceed: If the memory usage is below the threshold for the normal memory usage, no further action is taken. If the memory usage is higher, the next step is dependent on the program's processing speed state.
If the program is not currently confronted with speed problems, the garbage collector takes its time to check the age and number of incoming bytes of every entry in the hash table. It then deletes all the obsolete and small entries that are considered uninteresting, meaning the ones that do not have received more bytes than the according threshold during any of the observed intervals.
If the program executes its input at a rate more than a specified percentage below the input rate, the garbage collection tries to proceed faster through only considering entries of long

observation time when looking for entries with low number of incoming bytes. The idea is, that this way enough memory can be freed without checking every incoming bytes field of every entry.
If the program’s memory usage exceeds the second threshold for memory usage in terms of a percentage of the available memory, a more restrictive garbage collection is done: the in/out packet ratio is increased by a "tightening factor", given as a parameter, and all entries not holding data that fulfills this harder requirement for being worth observing, are deleted.
The last level of garbage collection comes into operation when the process runs out of memory eventually. In this case, a second garbage collector function is called that does not need to allocate new memory to do its work. This "no-memory garbage collection" just deletes entries from the hash table at random until it has freed the amount of bytes specified in the "free bytes" parameter in the program configuration.

When called at the end of a time interval, however, the garbage collection checks the in/out packet ratio and deletes all obsolete values, as in the stand alone version of the program.

6.3.4 Advanced log handler

For a theoretically endless execution of the plug-in, a maximal size for the output log has been introduced. To achieve a fixed total size of the log, the log is written to several files in a circular way. The maximal size per file and the number of files are set as parameters in the program configuration.

7 Results

7.1 Tests of the DDoS Detector plug-in on known attacks

The plug-in has been tested by replaying the NetFlow log files containing the known attacks discussed in Section 5 from the DDoSVax archive to UPFrame.
The known attacks have all been successfully detected by the DDoS Detector plug-in, along with some other events of high volume traffic to hosts that have not been reported as DoS attack victims.
The results of testing the plug-in on the known attacks are shown in the following subsections. The start and end of the detected attack are directly drawn into the graph of the off-line analysis. The times have to be interpreted as follows:

```
t1: Start of the attack
t2: Start of attack detection
t3: End of attack detection
t4: End of the attack
```

The parameters to define an attack were set as follows during these tests (see Appendix B):

```
packet_ratio_attack_threshold=100
packets_in_threshold=30000
bytes_in_threshold=50000000
tcp_short_pkt_ratio_threshold=15
short_tcp_flows_in_threshold=1000
```

7.1.1 Online analysis of the ICMP flooding attack (Figure 14)

Start of the attack: 23:06
Start of attack detection: 23:19
End of attack detection: 00:51
End of the attack: 00:51
Total number of bytes received during the attack, reported by the DoS Detector (for each of the attack victims): 3337002468

7.1.2 Online analysis of the TCP to myeth attack (Figure 15)

Start of the attack: 12:51
Start of attack detection: 12:53
End of attack detection: 13:04
End of the attack: 13:29
Total number of bytes received during the attack, reported by the DoS Detector: 1655014

7.1.3 Online analysis of the UDP to IRC server attack (Figure 16)

Start of the attack: 23:26
Start of attack detection: 23:37
End of attack detection: 23:44
End of the attack: 23:45
Total number of bytes received during the attack, reported by the DoS Detector: 5880443550

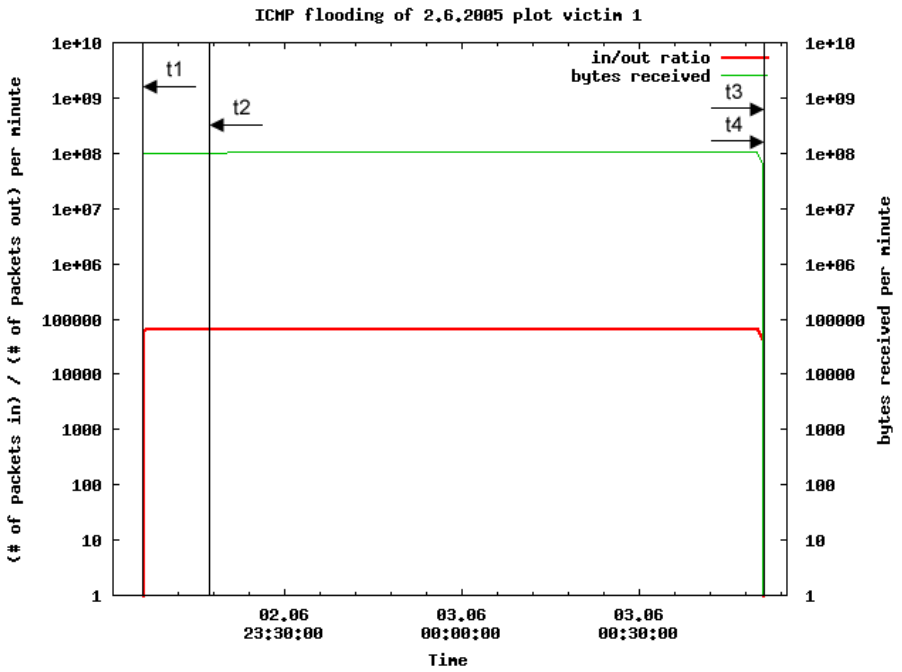

Figure 14: Online analysis of the ICMP flooding attack

7.2 Memory usage estimation

The resource usage of the plug-in is strongly dependent on its parameters. Running the plug-in with short observation period of 3, long observation period of 17, in/out packet ratio threshold of 15 and memory usage upper bound of 500 MB, the program used between 100 and 120 MB during the tests.

For an in/out packet ratio threshold of 15, the percentage of hosts with long observation time period is about 1 %.

Limitation of memory can result in delay of detection of attacks or failure to detect attacks, if the victim hosts have an in/out packet ratio that is only slightly above the threshold, since these entries could be discarded in favor of entries with higher ratios. The ICMP and UDP attacks discussed in this section, however, were still detected by the plug-in when run with a memory limit of 20 MB.

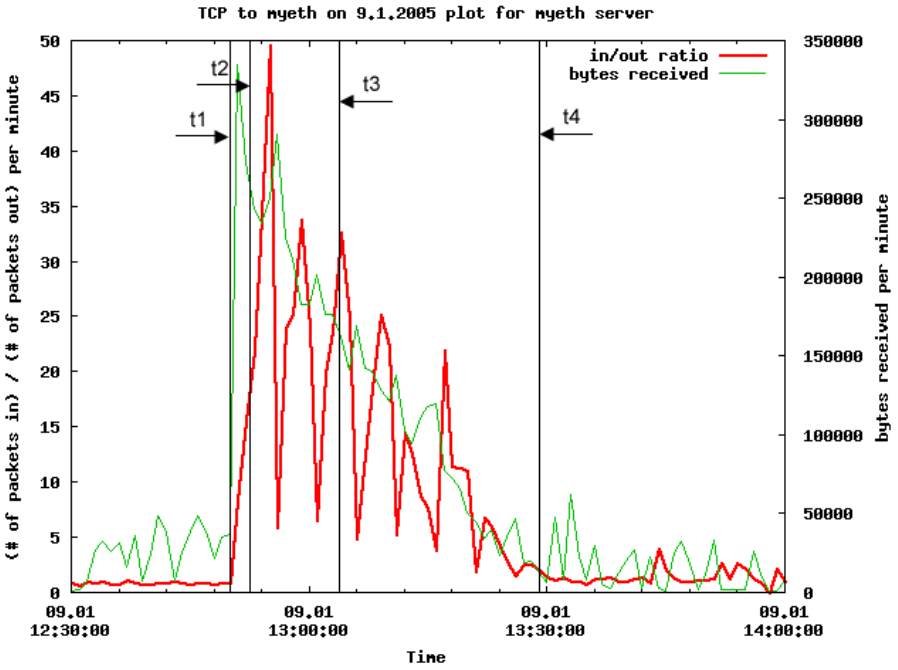

Figure 15: Online analysis of the TCP to myeth attack

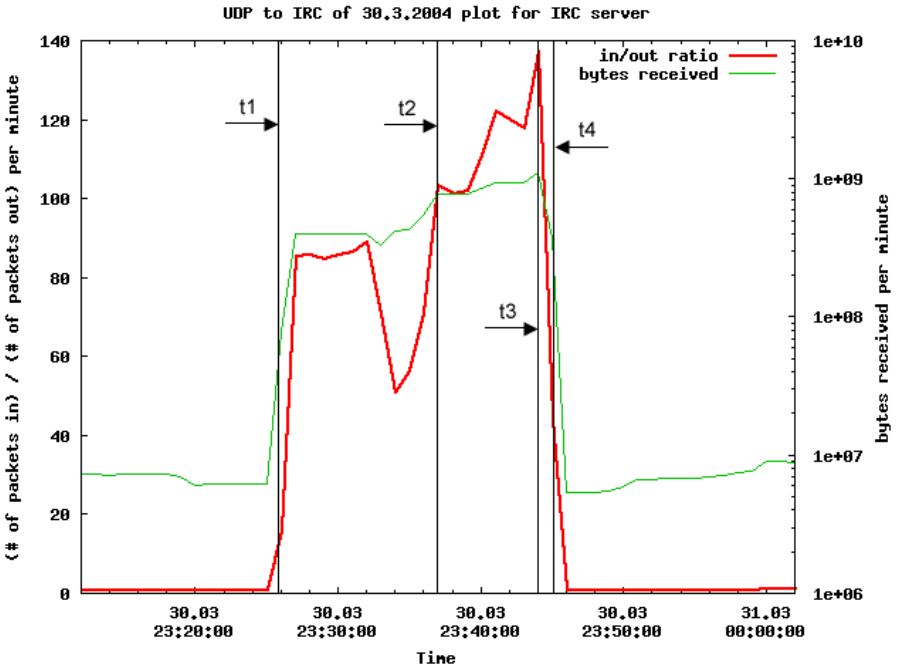

Figure 16: Online analysis of the UDP to IRC attack

8 Outlook

Proposals for further improvement of the algorithm are:

- Extension of the data structures to keep track of source addresses in order to reveal attack sources of non-distributed attacks.

- Enlargement of the data structures to store data for different protocols separately. This would improve the algorithm's ability to make appropriate decisions through a more differentiated view on the data.

- Inclusion of other algorithms when testing entries with high packet ratios to decrease the false positive rate.

9 Summary

The task of this diploma thesis was to design and implement an algorithm to detect DDoS attacks in the NetFlow logs of the SWITCH backbone border routers.
With taking into account the analysis of existing DoS detection algorithms and extensive off-line analyses of known attacks, an algorithm was designed to meet the requirements.
The algorithm has also been implemented in a real-time version as a plug-in for UPFrame, and successfully tested on archived data of known attacks from the past. The goals of this thesis were therefore fully achieved.

Major challenges met during this work were the study of related work with regard to the usability of proposed techniques for the DDoSVax environment, the handling of the vast amount of NetFlow data, that does not tolerate the slightest impreciseness, and the dealing with real-time problems arising with the design and implementation of the UPFrame plug-in.

10 Acknowledgements

I would like to thank my supervisors Thomas Dübendorfer and Arno Wagner for supporting me in my work on this thesis, and Lukas Ruf for providing the LaTeX template to write this report. Thanks also to Hans-Jörg Brundiers and the TIK Dienstgruppe for taking care of all the administrative issues related to the infrastructure at TIK.

A User's guide

A.1 DDoS Detector

A.1.1 DDoS Detector stand alone

The syntax to start the DDoS Detector in the stand alone version is the following:

```
./ddos_detector <options> <input file(s)>
Options:
   -i <interval in seconds> Default value is 60
   -s <subnet mask> Subnet mask in dot-notation. Default is 255.255.255.255
   -p <protocol> Name of protocol to observe
      (tcp,udp,icmp,www,nntp,ftp,ftpdata,smtp)
   -h Prints this message.
```

Table 1 explains the use of the different parameters.

Option	Argument type	Description
-i	Integer	Sets the time interval length in seconds
-s	IP address	Sets the subnet mask that will be added to host addresses. This option is used to make the detector observe a range of IP addresses instead of single hosts.
-p	String protocol name	Sets a protocol filter. If a protocol filter is applied, the program disregards all traffic except for the filtered protocol.
-h	None	Prints a list of the available command line arguments.

Table 1: Program arguments DDoS Detector stand alone

As an example, the program can be called as follows:
First write the input from both 19991* and 19993* log files into fifo pipes using `netflow_replay`:

```
./netflow_replay -f 19991_00020405_1116853199.dat.bz2 -n /tmp/tst1
./netflow_replay -f 19993_00020405_1116853200.dat.bz2 -n /tmp/tst2
```

Use `netflow_mix` to mix both streams into another pipe:

```
./netflow_mix -n /tmp/tst1 -m /tmp/tst2 > /tmp/input_pipe
```

Then run `ddos_detector` with this pipe as input file:

```
./ddos_detector -s 255.255.255.240 /tmp/input_pipe
```

A.1.2 DDoS Detector plugin

The syntax to start the DDoS Detector in the plug-in version is the following:

```
./detector-upframe-plugin <options>
Options:
   -c <config file name> If no file is specified, the program creates a new one
   -p <protocol> Name of protocol to observe
      (tcp,udp,icmp,www,nntp,ftp,ftpdata,smtp)
      with default parameters
   -h Prints this message.
```

Table 2 explains the use of the different parameters:

Option	Argument type	Description
-c	String	Configuration file name
-p	String protocol name	Sets a protocol filter. If a protocol filter is applied, the program disregards all traffic except for the filtered protocol.
-h	None	Prints a list of the available command line arguments.

Table 2: Program arguments DDoS Detector plug-in

To start the DDoS Detector plug-in using the default configuration file `ddos_detector.cfg`, however, there exist a start up script: `start_dos_detector.sh`.

A.2 Host Observer

The syntax to start the Host Observer is the following:

```
./host_observer <options> <input file(s)>
Options:
   -i <interval in seconds> Default value is 60
   -v <ip address> IP address of the host to observe (attack victim)
   -s <subnet mask> Subnet mask in dot-notation. Default is 255.255.255.255
   -p <protocol> Name of protocol to observe
      (tcp,udp,icmp,www,nntp,ftp,ftpdata,smtp)
   -h Prints this message.
```

The following table explains the use of the different parameters:

Option	Argument type	Description
-i	Integer	Sets the time interval length in seconds
-v	IP address	IP address of host to observe
-s	IP address	Sets the subnet mask that will be added to the observed host's address. This option is used to make the analyzer observe a range of IP addresses instead of single hosts.
-p	String protocol name	Sets a protocol filter. If a protocol filter is applied, the program disregards all traffic except for the filtered protocol.
-h	None	Prints a list of the available command line arguments.

Table 3: Program arguments Host Observer

See Section A.1.2 for an example of how to use `netflow_replay` and `netflow_mix` to run the Host Observer.

B Configuration Parameters

The parameters needed for the execution of the plug-in are collected in a configuration file. The name of the file is indicated to the program as an input argument. If no file is specified, the program creates a file called `ddos_detector.cfg` in the directory where the program was started and assigns default values to the parameters (see Table 4). Note, that the values of these parameters have a determining effect on the program's function.

Parameter	Type	Description	Default value
subnet_mask	IP addr	Sets the subnet mask that will be added to host addresses. This option is used to make the detector observe a range of IP addresses instead of single hosts.	255.255.255.255
interval_length	Integer	Time in milliseconds during which flows will be aggregated	60000
short_buffer_length	Integer	Number of time intervals for short observation period	3
long_buffer_length	Integer	Number of time intervals for long observation period. The value of this parameter should match the number of intervals that a NetFlow v5 record can span, i.e. (maximal length of NetFlow record / interval length) + 2	17
available_mem	Integer	Upper bound for dynamically allocated memory in bytes	200000000
normal_mem_usage	Float	Fraction of available memory whose usage is considered non critical (see Section 6.3.4)	0.8
high_mem_usage	Float	Fraction of available memory that when exceeded will result in tightening the packet ratio threshold at garbage collection (see Section 6.3.4)	0.95
free_bytes	Integer	Number of bytes that will be freed by non-memory garbage collection (see Section 6.3.4)	1000000
ratio_tightening_factor	Float	Tightening factor by which the packet ratio threshold will be multiplied for the restrictive garbage collection (see Section 6.3.4)	2.0

mem_check_time	Integer	Time between calls of the garbage collection in milliseconds. Assignment of a high value here may result in failure of the program to restrain memory usage to the given upper bound, since additional memory usage between calls is only estimated	1000
max_log_file_size	Integer	Maximal size of each log file in bytes	1000000
number_of_log_files	Integer	Maximal number of log files. The program overwrites log files in a circular way	20
short_duration_threshold	Integer	Length in milliseconds of TCP flows that are not counted as "short TCP flows" (see Section 6.2.5)	60000
small_packets_number	Integer	Upper bound for a "small" number of received packets. Host entries with a small byte and packet number are considered "small" (see Section 6.3.4)	1000
small_bytes_number	Integer	Upper bound for a "small" number of received bytes. Host entries with a small byte and packet number are considered "small" (see Section 6.3.4)	100000
packet_ratio_threshold	Integer	Ratio of received to sent packets that is considered unusually high. Host with a packet ratio above this threshold will be observed for the full possible length of flows	15
packet_ratio_attack_threshold	Integer	Minimum ratio of received to sent packets that will be logged as an attack	100
packets_in_threshold	Integer	Minimum number of received packet for an entry to be considered an attack victim	30000
bytes_in_threshold	Integer	Minimum number of received bytes for an entry to be considered an attack victim	50000000
tcp_short_pkt_ratio_threshold	Integer	Minimum ratio of incoming to outgoing short TCP flows that will be logged as a TCP attack	15
short_tcp_flows_in_threshold	Integer	Minimum number of incoming short TCP flows for an entry to be considered a TCP attack victim	1000

Table 4: Parameters DDoS Detector plug-in

C Original task description

The Problem

Denial of service (DoS) attacks exist in many varieties. They are an increasingly larger annoyance of the current Internet. However, still most DoS attacks go by unnoticed by network operators, especially if they are executed only over a short time period or not that massive. Only few tools and algorithms exist to reliably detect and analyse such attacks.

The Setting

In the context of the project DDoSVaX we collect and keep a long-term (>1 year) archive of flow-level Internet backbone traffic data. The DDoSVaX team has created the online processing framework UPFrame, which supports plugins for near-real time detection of Internet worm outbreaks and attacks. Several plugins were already developed and successfully validated.

The Task

By using current Internet flow-level traffic data and by replaying traffic data of earlier minor and major denial of service attacks from our NetFlow archive, the student will develop an algorithm and implement it as a UPFrame plugin that can be used for the early detection of (D)DoS attacks. The DoS attack detection algorithm will be validated against known attacks in our recorded data.

This task is split into the following subtasks:

Understand the NetFlow data and its processing via UPFrame

Before writing a plugin for the UPFrame it is crucial to get familiar with the NetFlow data and the UPFrame system. Existing plugins will be contemplated to get an idea of the complexity and the possibilities of a UPFrame plugin.

Develop an algorithm for (D)DoS attack detection

Taking into consideration existing algorithms for (D)DoS attack detection and earlier work on the subject within the DDoSVaX project, an algorithm will be developed aiming at a real-time detection of (D)DoS attacks by successively analysing NetFlow data.

Implement and test the algorithm

A design of the implementation of the detection algorithm will be developed. Then the program will be written for an offline analysis of NetFlow data. In the testing phase, the function of the program will be verified by processing archived NetFlow data of known DoS attacks of the past. As a last step, the algorithm will be implemented as a UPFrame plugin for real-time processing.

As a further optional component, the algorithm could be improved with the help of the test results.

Deliverables

During the work on this thesis the following deliverables are expected:

- Overview of related work on the subject: The relevant papers on the subject have to be described and categorised in a summary.

- Description of the algorithm to be implemented: Before the algorithm is implemented, it has to be presented along with performance estimations and a proposal of the software design.
- Real-time processing UPFrame plugin implementing the (D)DoS detection algorithm.
- Documentation: A written report will conclude this thesis.

Dates

This diploma thesis starts on June 6th, 2005 and will be finished on October 6th, 2005.

Supervisors

Thomas Dübendorfer, duebendorfer@tik.ee.ethz.ch, +41 44 632 71 96, ETZ G95
Arno Wagner, wagner@tik.ee.ethz.ch, +41 44 632 70 04, ETZ G95

References

[1] A. Akella, A. Bharambe, M. Reiter, and S. Seshan, *Detecting DDoS attacks on ISP networks*,
In *PODS Workshop on Management and Processing of Data Streams*, 2003.

[2] J. Baras, A. Carenas, V. Ramezani, *On-line detection of distributed attacks from space-time network flow patterns*,
In *Proc. 23rd Army Science Conf.*, 2002.

[3] P. Barford and D. Plonka, *Characteristics of Network Traffic Flow Anomalies*,
In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, 2001.

[4] P. Barford, J. Kline, D. Plonka, A. Ron, *A Signal Analysis of Network Traffic Anomalies*,
In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, 2002.

[5] R. B. Blažek, H. K., B. Rozovskii, and A. Tartakovsky, *A novel approach to detection of "denial-of-service" attacks via adaptive sequential and batch-sequantial change-point detection methods*,
IEEE Systems, Man and Cybernetics Information Assurance Workshop, June 2001.

[6] J. B. D. Cabrera, L. Lewis, X. Qin, W. Lee, R. K. Prasanth, B. Ravichandran, R. K. Mehra, *Proactive Detection of Distributed Denial of Service Attacks using MIB Traffic Variables - A Feasibility Study*,
In *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management*, Seattle, WA, May 2001.

[7] A. Challita, M. El Hassan, S. Maalouf, A. Zouheiry, *A Survey of DDoS Defense Mechanisms*, Department of Electrical and Computer Engineering American University of Beirut, 2004.

[8] R. K. C. Chang, *Defending against Flooding-Based Distributed Denial-of-Service Attacks: A Tutorial*,
IEEE Communications Magazine, October 2002.

[9] C.-M. Cheng, H.T. Kung, and K.-S. Tan, *Use of Spectral Analysis in Defense Against DoS Attacks*,
In *Proceedings of the IEEE GLOBECOM*, Taipei, Taiwan, 2002.

[10] K. Cho, R. Kaizaki and A. Kato, *Aguri: An Aggregation-based Traffic Profiler*,
In *Proceedings of QofIS2001*, 2001.

[11] F. Dressler, G. Münz, G. Carle, *Attack Detection using Cooperating Autonomous Detection Systems (CATS)*, Wilhelm-Schickard-Institute of Computer Science, Computer Networks and Internet, University of Tübingen, 2004.

[12] T. Dübendorfer, M. Bossardt, B. Plattner, *Adaptive Distributed Traffic Control Service for DDoS Attack Mitigation*,
In *1st International Workshop on Security in Systems and Networks (SSN 2005) hold in conjunction with IEEE IPDPS 2005 Conference, 19th International Parallel & Distributed Processing Symposium)*, Denver, Colorado, USA, April 4-8, 2005.

[13] L. Feinstein, D. Schnackenberg, R. Balupari and D. Kindred, *Statistical Approaches to DDoS Attack Detection and Response*,
In *Proceedings of the DARPA Information Survivability Conference and Exposition*, April 2003.

[14] P. Ferguson and D. Senie, *Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing*,
RFC 2827, May 2000.

[15] R. Gallati, *Near Real-Time Detection of Traffic Usage Rhythm Anomalies in the Backbone*, Diploma Thesis, TIK, ETH Zürich, 2005.

[16] J. Haggerty, Q. Shi, M. Merabti, *Beyond the perimeter: the need for early detection of Denial of Service Attacks*,
In *18th Annual Computer Security Applications Conference*, 2002.

[17] K. J. Houle, G. M. Weaver, *Trends in Denial of Service Attack Technology*,
CERT Coordination Center, October 2001.

[18] A. Hussain, J. Heidemann, and C. Papadopoulos, *A Framework for Classifying Denial of Service Attacks*,
SIGCOMM 2003, Karlsruhe, Germany.

[19] J. Ionnidis, S. M. Bellovin, *Implementing pushback: Router-based defense against DDoS attacks*. In *Proceedings of Network and Distributed System Security Symposium*, San Diego, CA, February 2002. The Internet Society.
In *Proceedings of the IEEE GLOBECOM*, Taipei, Taiwan, 2002.

[20] C. Jin, H. Wang, K. G. Shin, *Hop-Count Filtering: An Effective Defense Against Spoofed DDoS Traffic*,
In *Proceedings of the 10th ACM conference on Computer and Communications Security*, Washington D.C., USA, 2003.

[21] A.B. Kulkarni, S.F. Bush, S.C. Evans, *Detecting Distributed Denial-of-Service Attacks Using Kolmogorov Complexity Metrics*,
GE Research & Development Center, 2001.

[22] R. B. Lee, *Distributed Denial of Service: Taxonomies of Attacks, Tools and Countermeasures*,
Princeton University, Technical Report, 2004.

[23] X. Luo, R. K. C. Chang, *On a New Class of Pulsing Denial-of-Service Attacks and the Defense*,
In *Internet Society (ISOC), NDSS05*, 2005.

[24] J. Mirkovic, J. Martin, and P. Reiher, *A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms*,
http://www.lasr.cs.ucla.edu/ddos/ucla_tech_report_020018.pdf, 2002.

[25] J. Mirkovic, G. Prier, and P. Reiher, *Attacking DDoS at the Source*,
In *Proceedings of 10th IEEE International Conference on Network Protocols (ICNP 2002)*, Paris, France, 2002.

[26] D. Moore, G. Voelker, and S. Savage, *Inferring Internet Denial-of-Service Activity*,
Proc. 10th USENIX Sec. Symp.,2001.

[27] *NetFlow Services and Applications*, Cisco White Paper,
http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm

[28] C. Papadopoulos, R. Lindell, J. Mehringer, A. Hussain, R. Govindan, *COSSACK: Coordinated Suppression of Simultaneous Attacks*,
In *Proceedings of DARPA Information Survivability Conference and Exposition*, Washington DC, USA, 2003.

[29] V. Paxson, *An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks*,
ACM SIGCOMM Computer Communication Review, Volume 31, Issue 3, July 2001.

[30] T. Peng, C. Leckie, R. Kotagiri, *Detecting Reflector Attacks by Sharing Beliefs*,
In *Proceedings of the 8th Australasian Conference on Information Security and Privacy*, Wollongong, Australia, July, 2003.

[31] C. Siaterlis, B. Maglaris, P. Roris, *A novel approach for Distributed Denial of Service Detection Engine*,
NETMODE Lab Department of Electrical and Computer Engineering National Technical University of Athens, 2002-2003.

[32] V. A. Siris, F. Papagalou, *Application of Anomaly Detection Algorithms for Detecting SYN Flooding Attacks*,
In *Proc. of IEEE Globecom 2004 (Security and Network Management Symposium)*, Dallas, USA, November 2004.

[33] R. Sommer, A. Feldmann, *NetFlow: Information loss or win?*,
In *Proceedings of ACM SIGCOMM IMW 2002*, Marseilles, France, Nov. 2002.

[34] H. Wang, D. Zhang, K. G. Shin, *Detecting SYN Flooding Attacks*,
In *Proceedings of the IEEE Infocom*, pages 000-001, New York, NY, June 2002. IEEE.

[35] *UDP Port Denial-of-Service Attack*, Computer Emergency Response Team. CERT Advisory CA-1996-01, http://www.cert.org/advisories/CA-1996-01.html

[36] *TCP SYN Flooding and IP Spoofing Attacks*, Computer Emergency Response Team. CERT Advisory CA-1996.21, http://www.cert.org/advisories/CA-1996-21.html

[37] http://www.tik.ee.ethz.ch/∼ddosvax/upframe/ C. Schlegel, T. Dübendorfer, An extendible UDP processing framework (UPFrame)

[38] DDoSVax Homepage, http://www.tik.ee.ethz.ch/∼ddosvax/

[39] http://www.switch.ch/, The Swiss Research and Academic Network (SWITCH), AS559