TEKNILLINEN KORKEAKOULU
Tietoverkkolaboratorio

Friday, October 3, 2003

Author: Emmanuel Guiton

Supervisor: Prof. Jorma Jormakka

Instructor: Lic. Sc. (Tech.) Jarmo Mölsä

A Rate-Limiting System to Mitigate Denial of Service Attacks

Thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Engineering.

Standard Edition

Helsinki University of Technology	**ABSTRACT OF THE MASTER'S THESIS**		
Author:	Emmanuel Guiton		
Title of the Thesis:	A Rate-Limiting System to Mitigate Denial of Service Attacks		
Date:	10/03/2003	**Number of pages:**	12 + 97
Department:	Electrical and Communication Engineering		
Professorship:	S-38 Networking Technology		
Supervisor:	Professor Jorma Jormakka		
Instructor:	Licentiate of Science Jarmo Mölsä		

This document describes an implementation and the testing of an automatic defense system that uses rate-limiting to mitigate Denial of Service attacks.

Denial of Service attacks – and particularly the distributed ones - are amongst the latest and most problematic trends in network security threats. Currently, a few effective defense methods exist against them. In this document, the proposal is to jointly use the capabilities of attack detection (via Intrusion Detection Systems) and Quality of Service to rate-limit these attacks. As an automatic reaction, rate-limiting has an advantage over blocking: it preserves the legitimate traffic that is mis-identified as belonging to an attack.

This document describes in detail an already specified Rate-Limiting System. This system selects traffic into legitimate and attack aggregates thanks to an attack detection module. Based on this selection, routers direct the traffic aggregates into different queues. Attack queues are managed by a new Active Queue Management mechanism that enforces rate-limiting by randomly discarding packets.

This thesis presents mainly an implementation of the Rate-Limiting System in a Linux environment and its testing. It appeared from the tests that HTTP and FTP-downloading can handle one-way packet loss well, thus showing the suitability of rate-limiting to defend a website against low-bandwidth Denial of Service attacks such as typical TCP `SYN` or ICMP `Echo Request` flooding attacks.

Keywords: (Distributed) Denial of Service, Intrusion Detection Systems, Quality of Service, rate-limiting, Rate-Limiting System, RLS-AQM.

ACKNOWLEDGMENTS

This thesis has been carried out in the Networking Laboratory of Helsinki University of Technology. I would like to thank my supervisor, Prof. Jorma Jormakka, who gave me this opportunity. I also would like to thank him for his comments, which helped me to improve the quality of this work.

I would like to thank particularly Jarmo Mölsä, my instructor. I am grateful for all the guidance, comments, pieces of advice, and time he gave me since I started my job in the Networking Laboratory. I particularly appreciated the work we did together and on which most of this thesis is based.

Finally, I would like to thank all the people working in the Networking Laboratory. I'm thankful for the pleasant ambiance in which I could carry out my work.

Emmanuel Guiton
Espoo, Friday, October 3, 2003

TABLE OF CONTENTS

TABLE OF FIGURES

ACRONYMS

ACC:	As used in [Mah01]: *Aggregate-based Congestion Control*. Controlling network congestion by applying traffic-limiting on collections of packets sharing common characteristics (aggregates).
	Also: *Active Congestion Control* or *ACK Congestion Control*, not used here.
AF:	*Assured Forwarding*. A PHB group intended to improve the reliability of communications, using resource allocation and different drop precedences.
AQM:	*Active Queue Management*. Refers to mechanisms that provide network congestion indication to end systems suffering from it.
ATM:	*Automatic Teller Machine*. Cash withdrawal machine.
	Or: *Asynchronous Transfer Mode*. A connection-oriented, packet switched, and multiplexed network protocol that uses fixed length cells.
BGP:	*Border Gateway Protocol*. An exterior routing protocol used in the Internet to exchange routing information between Internet service providers' networks.
BW:	*Bandwidth*.
CBQ:	*Class-Based Queuing*. “a hierarchical class-based resource management [...] that can meet a range of services and link-sharing requirements” [Flo95].
CBS:	*Committed Burst Size*. Maximum amount of data that is allowed at committed rate in order for the end-system to be guaranteed that its packets are transmitted on a certain network.
CERT:	*Computer Emergency Response Team*. The CERT Coordination Center is an organization that provides exhaustive information about Internet security.
CIDDS:	*Common Intrusion Detection Director System*. A GOTS product also known as CID Director developed by the American Air Force. It is a dedicated hardware/software/operating system platform design to support intrusion detection tool projects

CIR:	*Committed Information Rate*. Guaranteed data rate an end-system gets to transmit its packets on a certain network.
CITRA:	*Cooperative Intrusion Traceback and Response Architecture*. A network architecture relying on the IDIP protocol to carry out traffic-limiting or blocking of DoS attacks.
DC:	*Discovery Coordinator*. As used in CITRA, a DC is a device with human oversight that controls and monitors activity throughout a CITRA community.
(D)DoS:	*(Distributed) Denial of Service*. Network security problem where an attacker aims to disrupt the normal operation of his targets' services. The attack is distributed when a (possibly large) set of compromised hosts is used to carry out the attack.
DSCP:	*Differentiated Services Codepoint*. A 6 bit value included in the Differentiated Service field in the IP header that is used to select a specific PHB.
EBS:	*Excess Burst Size*. Maximum amount of data over which a certain network does not guarantee anymore any transport service to an end-system.
EDF:	*Earlier Deadline First*. Scheduling principle that forwards packets in the order of the deadlines it first allocated to them.
EF:	*Expedited Forwarding*. A PHB group designed to ensure “a low loss, low latency, low jitter, assured bandwidth, end-to-end service through [Differentiated Service] domains” [Jac99].
EMERALD:	*Event Monitoring Enabling Responses to Anomalous Live Disturbances*. An intrusion detection product developed for research purposes by SRI International.
FIFO:	*First In First Out*. The most simple and common scheduling algorithm. The first packet to enter a buffer is the first packet that leaves it.
FTP:	*File Transfer Protocol*. User-level network protocol for file transfer between hosts.

GOTS:	*Government off-the-shelf*. Refers to a product developed for a government agency that controls totally the project, whether it is developed by an external entity or by the agency's technical staff.
HIDS:	*Host-based IDS*. cf NIDS.
HTTP:	*HyperText Transfer Protocol.* A generic, stateless, object-oriented application-layer protocol used to manage distributed, collaborative, hypermedia information.
ICMP:	*Internet Control Message Protocol*. Transport-layer protocol that provides IP-related message control and error reporting.
IDIP:	*Intruder Detection and Isolation Protocol.* A protocol aiming at coordinating DoS attacks reports amongst network devices and enabling automated responses.
IDS:	*Intrusion Detection System*. A system designed to detect intrusions by looking for abnormal activity patterns. It can be host-based (HIDS), network-based (NIDS) or an hybrid of both.
IETF:	*Internet Engineering Task Force*. An international organization that produces standard documents (RFCs) on the evolution and smooth operation of the Internet.
IIS:	*Internet Information Server/Services.* A web-server developed by Microsoft.
IP:	*Internet Protocol.* A connectionless network layer protocol designed to interconnect networks using packet-switched communications.
IRC:	*Internet Relay Chat.* Text based protocol for discussions between two or several people using the network media.
ISP:	*Internet Service Provider*. A company that provides individuals and companies access to the Internet and related services.
MPLS:	*Multiprotocol Label Switching.* A set of Internet standards for label switching that provide explicit control over network paths.

MSS:	*Maximum Segment Size*. The largest amount of data that a network device can handle in a single, non-fragmented piece.
MTP:	*Multicast Transport Protocol*. Reliable multicast transport protocol that delivers data on an one to many and many to many basis.
NAT:	*Network Address Translation*. An operation that consists in translating an IP address used within one network to a different IP address used within another network. NAT modifies the addresses of IP packets going from one network to another.
NIDS:	*Network-based IDS*. cf HIDS.
OFP:	*Optimized Flooding Protocol*. A routing protocol designed to carry out location discovery in wireless ad-hoc networks.
OS:	*Operating System*. The program that runs basic tasks and manages all other programs (applications) on a computer.
OSPF:	*Open Shortest Path First*. A link state interior routing protocol, widely used in the Internet, based on the Dijkstra algorithm (also known as open shortest path first).
PBS:	*Peak Burst Size*. Maximum amount of data that is allowed at peak rate so that the end-system is guaranteed that its packets are transmitted on a certain network.
PHB:	*Per-Hop Behavior*. In Differentiated Service networks, the externally observable forwarding treatment applied at a node to a traffic aggregate. A *PHB group* is a set of PHB sharing a same constraint.
PIR:	*Peak Information Rate*. Maximum data rate at which an end-system is assured to get its packets transmitted on a certain network.
QoS:	*Quality of Service*. Refers to the capability of a network to assure performance and to provide service differentiation to applications that use it.

RED:	*Random Early Detection.* An AQM and congestion avoidance mechanism that randomly drops packets on a increasing basis as network devices' queues get full.
RFC:	*Request For Comments.* A technical or organizational document about the Internet.
RLS:	*Rate-Limiting System.* Security system aiming to mitigate DoS attacks by limiting their traffic rates.
RMTP:	*Reliable Multicast Transport Protocol. A* reliable multicast transport protocol for the Internet. It provides sequenced, lossless delivery of data.
RSVP:	*Resource ReSerVation Protocol.* “A resource reservation setup protocol designed for an integrated services Internet. RSVP provides receiver-initiated setup of resource reservations for multicast or unicast data flows, with good scaling and robustness properties.” [Bra97]
RTT:	*Round Trip Time.* Elapsed time between the sending of a packet from a network device A to a device B, and the reception on A of a reply from B.
SLA:	*Service Level Agreement.* The SLA is an agreement between a customer and a network service provider that specifies the quality of the service that will be guaranteed to the customer.
SRI:	*Stanford Research Institute.* SRI International is a nonprofit research institute active in various fields including computer security.
srTCM:	*single rate Three Color Marker.* A meter that marks the packets from a flow according to its accordance with one specified traffic rate (CIR) and two burst sizes (CBS, EBS). Three levels of conformity (colors) are defined.
STAT:	*State Transition Analysis Tool.* A family of research intrusion detection tools that analyze transitions between different system state to detect intrusions.
TC:	*Traffic Control.* A program used to manage traffic control functions on Linux systems.

TCP:	*Transmission Control Protocol.* A highly reliable transport layer protocol designed for host-to-host communications.
trTCM:	*two rate Three Color Marker.* A meter that marks the packets from a flow according to its accordance with two specified traffic rates (CIR, PIR) and their associated burst sizes (CBS, PBS). Three levels of conformity (colors) are defined.
TFN:	*Tribe FloodNet.* A DDoS tool, used to coordinate and launch different kinds of DoS attacks against one or several particular targets from a high number of compromised hosts.
TOS:	*Type of Service.* An 8-bit field in an IP packet header used by upper layer protocols to specify how routing should be optimized for the packet (in terms of delay, cost, reliability, etc.).
T1:	The T-carrier service is a long-distance, digital communication line provided by a common carrier. Several capacity levels are defined including T1, which has a speed of 1.544 Mbps.
UDP:	*User Datagram Protocol.* An unreliable transport layer protocol designed for host-to-host, multicast, and broadcast communications.
WFQ:	*Weighted Fair Queuing.* A widely implemented class of scheduling algorithms that support bandwidth and delay bounds. It also refers to the original algorithm from which the WFQ class evolved.
WRR:	*Weighted Round Robin.* A scheduling mechanism that serves connections in a round after round fashion. During one round, a connection gets a proportion of service according to its pre-allocated weight.

1. INTRODUCTION

In the field of computer networks, security concerns arose as soon as the discipline was born. Since then, both attacks and defenses have pursued their evolution towards more and more sophisticated means. Regarding network attacks performed using remote computer tools, Denial of Service (DoS) attacks – and more particularly the distributed ones (DDoS) – are one of the latest and most powerful threats that have appeared. Currently, there is not any automatic and effective tool that exists to protect a system against this kind of attack. This document focuses on this problem and presents a defense mechanism that can mitigate DoS attacks.

The proposal is based on joining the capabilities of attack detection and traffic control to rate-limit attacks. Rate-limiting allows mitigating a flooding attack while preserving service quality for all legitimate traffic [Möl-1] – what blocking cannot guarantee. Two architectures already have been proposed to achieve similar goals: the Aggregate-based Congestion Control (ACC) [Mah01] and the Cooperative Intrusion Traceback and Response Architecture (CITRA) [Sch01]. However, these two propositions did not include any tests focused on the usability of a legitimate application whose traffic is mis-identified as attack traffic. Yet, the bounds of the usability of rate-limiting can be determined by studying this case. Thus, the main purpose of this document is to describe an implementation of a rate-limiting system and to present the results of some tests focused on its usability. This work, described in [Möl-2], has validated the relevance of using rate-limiting as a defense mechanism against DoS attacks. Indeed, the system proved to be useful to mitigate low-bandwidth flooding attacks.

The first part of this document is a survey that introduces the topics related to this work. First, it presents the Denial of Service, which is the security issue this study addresses. Some well-known attack mechanisms and DoS tools are analyzed. Then, the Intrusion Detection System (IDS) and the Quality of Service (QoS) technologies are depicted. The proposal described in the second half of this thesis relies on the association of these two methods. The first one provides DoS attack detection, the second one provides the mechanisms to rate-limit these attacks. In a last step, the current defense mechanisms against DoS attacks are reviewed and the needs this study addresses are presented. The second half of this thesis focuses on the design of a Rate-Limiting System (RLS) and its tests. The theoretical requirements, as published in [Möl-1], are first presented. Then a simple implementation of the RLS for a Linux environment is described. This implementation was tested in a small network and the results and their analysis conclude this document, along with some issues that should be addressed to improve the system.

2. DENIAL OF SERVICE ATTACKS

This chapter describes the security problem that this document addresses: Denial of Service. First, some general information about this security issue is introduced and then the subject is deepened with the description and analysis of some attack methods and tools.

2.1. Introduction to the Denial of Service

2.1.1. A brief history

From private individuals saving personal documents to defense departments storing top secret intelligence reports, computers are often used to store sensitive data. Networks, for example banks' Automatic Teller Machine (ATM) networks, are widely used to access and carry confidential information. The existence of certain firms like the .com companies relies solely on computer networks. Obviously, there is a need for privacy and security in these two examples. No one wants an unauthorized third party to look at or modify sensitive data. Commercial companies cannot accept threats that can put their health, reputation, or even existence in danger. As omnipresent and sometimes critical tools in people's life in developed countries, computer related technologies need to meet these security requirements.

In fact, security is quite an old concern in the computer technology field. As early as the late 1950's, computers included mechanisms to ensure that programs could not use someone else's disk space. During the 1960's, several security methods (such as controlling access to files or protecting passwords by encryption), whose principles are still in use today, were developed. The growing trend led computer security to be studied as a full discipline during the beginning of the 1970's. Since then, new security issues have appeared as fast as old ones were solved. As research teams were developing new defense mechanisms, the underground attack field was also maturing and producing more and more sophisticated tools, raising new problems. The advent of the Internet particularly gave a boost to the importance of computer security. While often used as a business media, the Internet is a highly non-secure, non-trustworthy environment from a security point of view. Whit a few resources, malicious people are offered a world wide operation scale.

Moreover, offenders are most often protected by a legal vacuum in many countries and by a lack of international legal cooperation.

Events on November 2nd, 1988 marked the appearance of large scale attacks made possible by the growth of the Internet. Robert T. Morris, an American doctoral student, created a malicious program called “Internet Worm” and launched it on the Internet. A *worm* is a self-propagating malicious program that infects computers on a network. It is different from a *virus* that cannot be activated nor propagate itself without human intervention. Using known security problems in a particular system, Morris' worm infected about three to six thousand stations (5% to 10% of the whole Internet at that time) and caused a severe service disruption [Rey89]. This was the first large-scale attack on the Internet and it can also be classified as the first *Denial of Service* (DoS) attack. The following section focuses on this particular kind of threat.

2.1.2. Principles of the Denial of Service

At the end of the 1990's, DoS attacks became very popular amongst the *cracker* communities. This was due to the appearance of automated DoS attack tools on the Internet that made attack process within inexperienced crackers' capabilities. In this document, the term cracker refers to “a person who compromises a computer security system without permission of the person operating the system” [Wik03-1]. The word *hacker* is often used with the same meaning while it originally describes “someone who knows a (sometimes specified) set of programming interfaces well enough to write novel and useful software without conscious thought on a good day” [Wik03-2].

The goal of DoS attacks is to deny users of a system access to their legitimate services. In computer networks, examples of such services are sending e-mails, surfing on Internet or downloading a file. From a more technical point of view, network services are applications whose operations rely on communications over a network. End-user applications typically use a particular transport layer protocol and a particular port number. Some common applications use official, well-known numbers that are in the range 0-255 [Rey94]. For example, Telnet (terminal emulation program) uses TCP on port 23, FTP (file transfer) uses TCP or UDP on port 21, IRC (text based chat) uses TCP or UDP on port 194, etc.

DoS attacks can be classified in several kinds, all of them are not studied here. A first way to deny services is to physically damage the target. This document does not address this kind of issue, it only concentrates on remote attacks operated through a network using computer tools. Network attacks can be referred to as *flooding* attacks or *logic* attacks. Currently, the first ones are the most used. They involve consuming all of the target's available resources, which is one of the most obvious way to carry out Denial of Service. Indeed, every system has intrinsically a finite and consumable amount of resources. For example, a low capacity network link can be overloaded with useless messages, making normal network communications impossible through this link. An attacker can also feed a processor with heavy calculations, exhausting its resources so other applications cannot be served. As for the logic attacks, they rely on an intelligent exploitation of vulnerabilities in the target [Moo01]. They usually use some known security vulnerabilities in a way that it seriously damages the target, bringing it down in the worst case. The exploitation of certain software bugs can lead an attacker to get the administrator privileges and to be able to execute any malicious program.

With time, DoS attacks became more and more refined. In 1999, a new kind of DoS attack involving large sets of computers appeared: the *Distributed Denial of Service Attacks* (DDoS). The next section provides a more detailed description of them.

2.1.3. Distributed Denial of Service Attacks

2.1.3.1. Definition

The first publicly reported DDoS attacks appeared in the late 1999. These attacks quickly became increasingly popular as communities of crackers developed and released automated tools to carry them out.

A DDoS attack has the same goal as a normal DoS attack: to disrupt a service, to deny legitimate users their access to some service. However, while a DoS attack only involves one attacker, a DDoS attack uses numerous coordinated hosts to carry out the attack. That is why it is called distributed. The real attacker(s) command(s) a set of compromised hosts to make them execute an attack against a single target (or a few targets), thus multiplying the effectiveness of the attack.

Thanks to its distributed characteristic, a DDoS attack achieves much better effectiveness than a simple DoS attack when aiming to consume target system's resources. It also enables a diversity of sources that makes it more difficult to differentiate from legitimate traffic: no easily identifiable attack *traffic aggregate* appears. An aggregate is a set of *traffic flows* that share the same forwarding characteristics and the same link resources. A flow refers to the whole set of the packets traveling from the same source to the same destination in a network. Granularity of source and destination can vary from a precise application in a host to a subset of the network.

2.1.3.2. How crackers operate

Typical DDoS attacks that use dedicated tools usually involve two steps. First, a network of compromised hosts is deployed. Then, this *DDoS network* is used to launch the attack. As described in the following 2.2. section, some other methods can also be used to perform DDoS attacks.

To build the DDoS network, groups of crackers have to go through a mass-intrusion phase. However, dedicated DDoS tools do not generally include automated mechanisms for spreading and compromising hosts. Therefore, in the earliest important attacks, crackers had spent weeks installing the DDoS tools manually on systems they had compromised by exploiting known weaknesses like the *buffer overflow* vulnerability. This kind of attack tries to store more data in an application buffer than it was designed to hold. Data exceeding the buffer capacity overflow into adjacent memory areas. This can directly cause damages or it can be used to install bad instructions in the executable stack. Typically, attackers use buffer overflow attacks to take control over the target station. They use long strings whose ends contain malicious executable code that is then run by the operating system once the original function ends and the next operation can be carried out.

Later, the deployment process has been more and more automated, using worms or more specific deployment tools [Möl-1]. An example of a deployment tool is T0rnkit [CER00-3]. Several versions of this program exist and include different functionalities. One uses scripts to scan for weak systems, exploits their vulnerabilities, and installs DDoS programs. The number of the victims of the deployment phase can range from several hundreds to several thousands of hosts. A DDoS network is usually organized according to a three-level top-down hierarchy (see figure 1 hereafter). On the top, attackers controls a small number of *handlers* that themselves control a high number of *agents* (according to the terminology defined by the Computer Emergency Response Team (CERT)). Agents are the final devices that effectively carry out the attack.

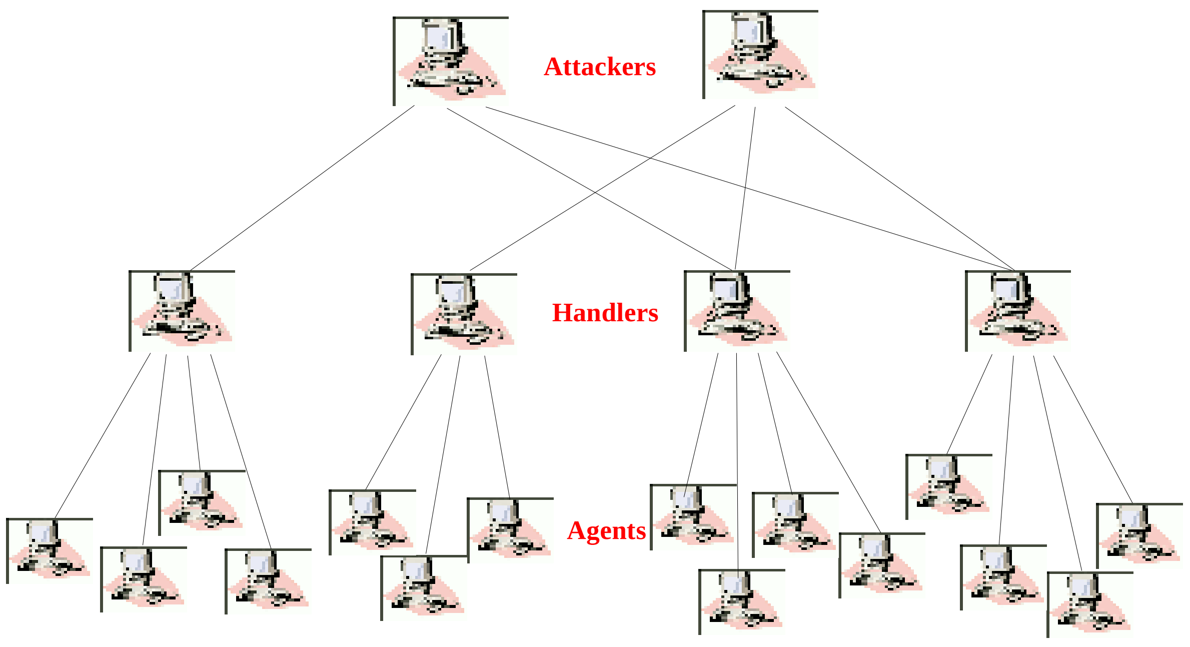

Figure 1: A common DDoS network architecture

Once the DDoS network is operational, attackers can proceed with the real attack. Every cracker who participated in establishing the DDoS network may not take part in the actual attack. First, at least one attacker sends orders to one or several handlers. Depending on the DDoS tool, commands generally specify the targets and a particular DoS attack mechanism if a choice is available. The handlers then forward the instructions to the agents that finally execute the attack. Only a small delay occurs between the launching of the attack and the execution itself by any agent. Thus, all the agents start to generate attack traffic almost simultaneously. This synchronization maximizes the attacks' effects by creating the strongest possible flood at a time. Various protocols are used to communicate between parties, and communication means are very tool-dependent. At first, the most widely used protocols were ICMP, UDP, TCP, but from August 2000, IRC began to be adopted as the basis for communications between attackers and handlers [Hou01].

2.1.4. Effects of DoS Attacks

From time to time, news reporting a DoS attack against a well known website appear on the Internet. Sometimes, events are even published in newspapers when they have been particularly harmful. One example is the wave of DDoS attacks on the beginning of February, 2000. These attacks became quite famous due to their effectiveness [Gar00]. On the 6th day, the portal of Yahoo! Inc. was shut down for three hours. The next day, Buy.com Inc, eBay, Amazon.com, and CNN websites suffered from the same attack. The series ended on the 8th with the websites of ZDNet, E*Trade, and Excite. To summarize, eight high-profile commercial websites were inundated with traffic rates up to 1 gigabit per second.

DoS attacks can also directly affect every Internet user by causing instabilities in the backbone network. It has been shown that the deployment phases of the Code Red and Nimda worms during the summer of 2001 increased the BGP advertisement rate in some important backbone networks by 8 and 25 times respectively [Cow01]. The *Border Gateway Protocol* is used in the Internet to exchange routing information between Internet service providers' networks. The worms' propagation caused failures in network reachability and BGP routers' mechanisms.

The first obvious consequence of DoS attacks is service disruption. However, while for most Internet users these attacks are mainly a temporary annoyance, it takes a much greater importance for commercial companies. Every company involved in an attack may lose reputation, the victims (like Yahoo! Inc., eBay, etc.) as well as their providers. The Internet Service Provider will be blamed for letting attacks occur on its network. Companies manufacturing the hardware or software components will suffer from bad publicities as their products contain the security failures that enable the attacks. Microsoft is one well-known brand that suffers from bad publicity about its products' quality, particularly because of security vulnerabilities that were discovered in its pieces of software. Although it is very difficult to assess, companies also directly lose money if they do some Internet related business like on-line sales. An interruption of the e-commerce service is a loss of profit. To protect themselves, companies have to spend additional money in risk management (insurances, security improvement of the corporate network, etc.). Finally, some network service providers operating in the Internet backbone may directly see their revenue expenditures increase because of the additional traffic load that DoS attack generate in the network equipments.

2.2. DoS Methods and Tools

2.2.1. DoS Attack Mechanisms

This section presents several well-known mechanisms that are often used by the main (D) DoS tools to disrupt services.

2.2.1.1. Spoofing addresses

Basically, spoofing an address only means changing its value for a wrong one. This is a quite simple and often used technique. By spoofing source IP addresses, an attacker can mislead its target's replies, making it try to communicate with some random or even non-existent hosts. This misbehavior serves as a basis for attacks such as the TCP SYN flooding attack described hereafter. It can also be used to make an attack harder to be recognize. For example, you do not expect a single host to use 1000 connections to the same server at once, but 1000 different hosts accessing the same server can look totally legitimate.

If no exhaustive solution can address this problem, there is a simple and well-known – but unfortunately not always implemented – method to easily reduce the number of spoofed packets [Fer00]. Basically, a router interfaces two or more networks whose address spaces it knows. Packets coming from a certain network cannot have a source address that is not part of this network's address space. Such packets with addresses outside from the correct range must be discarded because they are either erroneous or spoofed. Recording this information can then help to identify the true source of an attack.

2.2.1.2. UDP flooding

UDP flooding attacks are a problem known since 1996 but it is still topical. In 2000, DDoS tools using this method (for example Trin00, TFN) have been successfully used. The UDP flooding attack aims computers providing external UDP services. Common targets are services named echo (sends back received messages) on port 7, daytime (provides date and time in a human readable ASCII string) on port 13, time (provides machine readable date and time) on port 37, and chargen (character generator) on port 19. If two UDP services are connected and produce traffic, they can deliver an excessively high number of packets causing Denial of Service [CER97]. This attack can be either operated locally between two services on a same machine or between two distant stations. In the latter case, the network that links the two hosts can also suffer high congestion and prevent other stations on the same network to use it. One strength of this attack is that the attacker does not need to gain unauthorized access: anyone having network connectivity can launch it.

There is no perfect solution to defeat UDP flooding attacks. Recommendations particularly include disabling useless UDP services and blocking UDP ports at firewalls. However, if you really need to provide these services, then you cannot protect them. You need to monitor activity on the UDP ports for signs of misuse (i.e. you should run an intrusion detection system).

2.2.1.3. TCP `SYN` flooding

Every TCP connection begins with a *three-way handshake*. The `SYN` message is the first message a host wishing to establish a TCP connection (client) sends to its correspondent (server). It is simply used to request a new connection. When the server can accept the connection, it replies with a `SYN-ACK` message and waits for a third message and final acknowledgment (`ACK` message) from the client. A connection is described as *half-open* when a server has sent a `SYN-ACK` message but not yet received the corresponding `ACK` message.

The goal of a TCP `SYN` flooding attack [CER00-4] is to overload a server by sending it a high number of TCP `SYN` messages with spoofed source addresses. More generally, *flooding* refers to situations when a network resource is overloaded by messages. In this particular case, the server replies to `SYN` messages and waits for `ACK` messages that will (most often) never arrive as the originating client does not exist (or has not requested the connection if by chance the spoofed source address points to a real host). As the server can only accept a limited number of connections, the attack quickly consumes them all and triggers the rejection of other connection attempts. Legitimate users can then no longer use TCP services on the victim server. Operating systems are more or less vulnerable to this threat according to the length of the timeout after which a pending connection is canceled.

Systems can eventually recover thanks to this timeout mechanism, but the attack usually requests new connections faster than half-open connections expire. Moreover, in some cases, the server can also exhaust memory, crash, or be rendered totally inoperative.

The first piece of advice to face this attack is to implement filtering rules to discard spoofed IP packets as described above in 2.2.1.1. A host can succeed in handling an attack correctly (i.e. if the attack does not prevent the target to function). In that case, a person using locally the host may not be able to notice the attack as the station will still be able to establish outgoing connections. Then, detecting the attack requires checking the state of the server's network traffic. An unusually high number of connections in the SYN_RECEIVED state may indicate that the system is under attack.

2.2.1.4. ICMP flooding and amplifying techniques

ICMP is a protocol used “to provide feedback about problems in the communication environment” [Pos81]. For example, it delivers information concerning network errors or congestion, it helps to troubleshoot, and it announces IP packet timeouts. ICMP allows checking if a host on a network is responding by sending to it an ICMP `Echo Request` message. If the target station receives this message, it sends a ICMP `Echo Reply` back to the sender. ping is a popular program that uses this feature to measure the response time of remote hosts.

The ICMP flooding attack aims to uselessly consume the network resources of the victim. The goal is to overwhelm a target with ICMP messages to which the host will respond, losing processing time and wasting network bandwidth. Fragmenting or rearranging IP packets can also make the attack more effective. These operations require more processing time to analyze the incoming packets, as the station has to correctly rebuild them first.

Amplified attacks generally use broadcast IP addresses to increase the effect of an attack. Broadcast IP addresses are used to send messages to all the hosts connected on a particular network. They are usually formed by setting the bits of their host part to 1 (like 10.255.255.255). By sending a message to a broadcast address, an attacker ensures that a message will be received by several (and potentially many) hosts. If this message requires a response, then the attacker can generate a (potentially) high number of messages as all the hosts will reply. However, the tendency of each station to answer is highly dependent on its configuration. For security purposes, some organizations scan for addresses of vulnerable networks and published them in websites such as http://www.powertech.no/smurf/. Thus, the information is also available for any crackers that then know which broadcast addresses can be effectively used in a future attack.

The same kind of amplifying mechanism can also rely on an unexpected behavior of an operating system. For example, it has been noticed that an intruder could abuse Apple's operating system MacOS 9 to generate a high volume of traffic in response to a small amount of traffic [Cop99]. It was noticed that some unusual 29 byte long UDP packets were triggering replies with 1500 byte long ICMP packet. This asymmetric property has been successfully used by a test tool that made MacOS 9 stations send 1500 byte long ICMP messages in reply to 40 byte long UDP messages. Therefore, this tool achieved a 37.5 time amplification. To be effective, this attack has to use Mac OS 9 stations that are connected to the Internet via high bandwidth networks like T1 networks. Otherwise the capacity of the network would act as a bottleneck.

The smurf attack [CER00-1] is an example of DDoS attack using both ICMP flooding and amplifying techniques. It involves three parties: the attacker, some intermediary stations (potentially victims too), and the victim. The attacker sends ICMP `Echo Requests` with a spoofed source IP address to some broadcast destination addresses. All the stations reached by an ICMP `Echo Request` send back an ICMP `Echo Reply` to the spoofed source address, which is in fact the address of the victim. As a result, the victim is overloaded with ICMP `Echo Reply` messages, causing severe network congestion or outages.

At the intermediaries level, there are ways to solve the problems. Apple released a patch that fixes the asymmetry bug in MacOS 9. To avoid smurf attacks, some simple setting modifications are needed: denying IP broadcast messages coming from outside networks, disabling responses to ICMP requests sent to broadcast addresses. Although this functionality can often be disabled (some legitimate applications use it, however), there will always be unaware network administrators who will not do so when possible. Unfortunately for victims, there is no easy solution to protect a network from flooding. The most effective actions are to alert the Internet Service Provider and the intermediaries so they can temporarily block the traffic or modify their network configuration.

2.2.2. Examples of DDoS Tools

This section does not provide an exhaustive overview of DDoS tools. It is intended to illustrate the problem by introducing some concrete examples and describing the way the tools work.

2.2.2.1. The Code Red worms

The Code Red [CER02] worm exploits a known buffer overflow vulnerability in Microsoft IIS servers. First, the worm looks for a web-server by trying to connect to TCP port 80 (the HTTP service) on a randomly chosen address. When succeeding, it tries to exploit the buffer overflow vulnerability. If the victim has this vulnerability, the worm usually succeeds in executing itself on the host. Otherwise, on some systems, it can lead the IIS servers to crash or to stop forwarding packets. Once installed, the worm performs three different actions according to the day of the month. Between days 1 and 19 it will try to propagate itself. Between days 20 and 27 it will launch a flooding DoS attack against a particular fixed address. For the rest of the month, the worm just sleeps; it stays in an idle state.

The first effect of this attack is the possible defacement of the web-pages hosted by the compromised host. A second problem comes from a side effect of the spreading technique. To propagate itself, the worm scans intensively random addresses. This causes a performance degradation on the infected station. This degradation can be severe since the worm can infect a station multiple times so several instances of the worm can scan for IP addresses at the same time. In the first version, each instances of the Code Red worm were using the same random number generator seed to create the list of IP addresses to be scanned. Therefore all infected hosts were scanning the same IP addresses, which increased the severity of the Denial of Service on the corresponding hosts. Although Code Red is not precisely categorized as a DDoS tool, in facts it is one. Finally, as specified above, it can execute a flooding DoS attack. This attack is distributed because Code Red spreads widely, creating a large pool of attack sources.

Another worm, called Code Red II [CER01-2], compromises computers by exploiting the same buffer overflow vulnerability in Microsoft IIS service. The rest of its behavior is totally different. The worm only infects once: when executed, it first looks for the

CodeRedII *atom* (a unique identifier on Microsoft systems). If the atoms exists, it enters in sleeping mode for ever, otherwise it creates the atom and continues its actions. The second action is to check the default language of the system. If it is set to Chinese, the worm starts 600 threads to scan during 48 hours, otherwise it starts half of it for 24 hours. Then, it places the CMD.EXE binary file in a publicly available directory, allowing an intruder to run commands with the IIS server process' privileges. Finally, it creates a Trojan horse copy of explorer.exe and makes the C: and D: drives vulnerable. The Trojan horse also calls the genuine explorer program to hide itself.

The main purpose of the Code Red II worm is not Denial of Service. However, it can be used as an automated mechanism to deploy DDoS tools. Once it has infected a system, the worm has the necessary privileges to install any attack program. An intruder can also alter or delete files, which may result in Denial of Service for applications relying on these files. Finally, as for Code Red, the scanning activity of the worm may result in a de facto DoS attack.

2.2.2.2. Dedicated DDoS tools

Trin00 [CER01-1][Dit99] is explicitly categorized as a DDoS tool. It coordinates UDP flooding DoS attacks from many sources, using the DDoS network architecture presented in figure 1. In Trin00 terminology, handlers are called masters and agents are referred to as daemons. In a first step, a DDoS network is built using the spreading and compromising techniques presented above. Once the Trin00 network is operational, crackers launch DDoS attacks by connecting to one handler and providing it one or several addresses to flood with UDP datagrams (see subsection 2.2.1.2. above). The handler then passes the instructions to the agents, which execute the attack during a specified period of time. Trin00 source code includes a shell that the attackers use to remotely control the handlers. The connections between attackers and handlers are set-up using TCP and are protected by a password. However, communications are sent in plain-text format, making Trin00 vulnerable to standard TCP attacks. Communications between handlers and agents use UDP. A password is required to establish the connections, which are protected by encryption.

The Tribe Flood Network (TFN) DDoS tool [CER01] [Dit99-1] works much like Trin00. It also uses the classic DDoS network architecture, the same way Trin00 does. However, it does not perform only UDP flooding attacks but it also uses TCP `SYN` flooding, ICMP `Echo Request` flooding, and smurf attacks. TFN can generate packets with altered size and random source IP address and port. All these mechanisms make the detection of DDoS traffic amongst legitimate traffic much harder. Communications between the attacker and the handlers are based on the client/server model and can use any of the TCP, UDP or ICMP protocol. TFN also uses ICMP instead of UDP for the communications between handlers and agents. No passwords nor encryption protects TFN's communications.

TFN2K (standing for Tribe Flood Network 2K) [CER00] can be seen as an improved version of TFN. It has the same basic properties and in addition it implements features to make it “difficult to recognize and filter, to remotely execute commands, to obfuscate the true source of the traffic, to transport TFN2K traffic over multiple transport protocols including UDP, TCP, and ICMP, and features to confuse attempts to locate other nodes in a TFN2k network”[CER00]. The true source of attacks is also masked by spoofing IP addresses. If ever a network uses ingress filtering (see above in section 2.2.2.1. Spoofing addresses), TFN2K is capable of generating addresses that seem to belong to this network. Finally, TFN2K does not only perform traffic flooding attacks but it can send malformed or invalid packets aiming to crash or introduce instabilities in systems.

Stacheldraht (German for “barbed wire”) [Dit99-2] is a program based on the original TFN's source code combined with features from Trin00. Stacheldraht has probably been written from early versions of Trin00 as it does not include the “on demand” root shell, a feature also absent in the first versions of Trin00. It uses the same architecture, with the attacker commanding a set of handlers, each of them commanding a large set of agents (upper bound is set to 1000 in the source code). It also uses TFN's DoS attack mechanisms. Unlike Trin00 and TFN, communications between the attacker and the handlers are encrypted, thus protecting them against TCP session hijacking. Communications between handlers and agents use ICMP. They are also encrypted (using the Blowfish algorithm) and a password is required to connect to a handler. Stacheldraht is able to upgrade the agents on demand, a features that neither Trin00 nor TFN have. New copies of the program are stored in stolen accounts at some website. Upon request, the agents delete their current image, download the new copy, start running the new image, and finally exit. Beforehand, Stacheldraht agents check if the networks on which they are running allow spoofed IP addresses. From this test, they learn if they can spoof only the last octet of the IP addresses or all the four octets.

3. INTRUSION DETECTION AND QUALITY OF SERVICE TECHNOLOGIES

This chapter presents two technologies that are not specifically addressing the Denial of Service problem: the Intrusion Detection and Quality of Service fields. However, both of them include capabilities that can be jointly used to create a system that mitigates DoS attacks. Before such a system is described, the required knowledge about these two technologies is described hereafter.

3.1. Intrusion Detection Systems (IDSs)

Intrusion Detection Systems [All00] are meant to detect any abnormal (and malicious) behavior in a system. Their use is particularly relevant to detect DoS attacks. This chapter will describe the general principles of IDSs and then their applications to defend a system against DoS attacks.

3.1.1. Overview of IDSs

3.1.1.1. Description of IDSs

In the computer security field, an *intrusion* denotes a successful attack aiming at penetrating inside a system in an unauthorized manner. However, the phrase “intrusion detection system” is misleading because it refers to the detection of both successful and unsuccessful attacks. An IDS is a piece of software that aims to detect any attack attempt against the system it defends. It can be basically divided into three components: *sensors*, *analyzers*, and a user interface [All00].

Sensors are responsible for two tasks: collecting data that can contain evidence of an intrusion and forwarding these data to analyzers. Depending on the IDS type (see section 3.1.2. hereafter), the data source can take various forms: raw network packets, operating systems log files, *honeypots*, and even outputs from other IDSs. Honeypots are controlled environments designed to appear to intruders as systems with known vulnerabilities. In fact, they are able to detect intrusions and preserve a known state. Intruders' actions can then be easily tracked and analyzed.

Analyzers have to determine from the collected data if any intrusion has occurred. Two different analysis approaches are possible: the *misuse* or the *anomaly detection.* The attack misuse detection (or *attack signature detection*) rely on a prior knowledge of possible intrusions. These attacks are identified by characteristic patterns, called *attack* signatures, created from data such as IP addresses, port numbers, used protocol, presence of a precise string in the payload, etc. Anomaly detection operates in the opposite way: IDS have knowledge of the expected, legitimate behavior of the system. In that case, an intrusion is detected when an IDS notices any activity that does not match any predefined legitimate behavior. Analyzers may produce alerts upon detection of intrusions and may also include evidence of them. Certain IDSs can even automatically trigger some actions against the detected attacks. Currently, possible actions are not very varied and the most common one is to block the attack traffic.

Finally, the user interface enables a user to control the behavior of an IDS or to look at output from it. Human operations are needed to carry out operations that an IDS cannot automatically perform. For example, system administrators have often to (re-)configure an IDS or to act against an attacker upon detection of an intrusion. Looking at outputs from an IDS also enables to check the validity of its behavior. It can also be used to find evidences after a successful attack.

3.1.1.2. Why do we need IDSs?

Protecting a system cannot rely on a single technique. Many various components – such as firewalls, access control and authentication mechanisms, anti-virus software, IDSs – are needed to implement a good security architecture. Firewalls have proved to be useful filters at systems boundaries, but they cannot successfully stop every kind of attack. For example, they are inefficient when an attack is launched from inside a system (a legitimate user willing to perform unauthorized actions). They generally do not protect neither from unsecured modems nor from malicious mobile code. In addition, they often suffer from misconfiguration or bad *security policy* (the set of rules that govern the security requirements and specifications of an entity, like a corporation, for example). IDSs are currently seen as the necessary complement to firewalls because they can detect malicious activities that are transparent to firewalls.

To summarize, “intrusion detection is considered by many to be the logical complement to network firewalls, extending the security management capabilities of system administrators to include security audit, monitoring, attack recognition, and response.” [All00]

3.1.2. The Different Kinds of IDSs

Based on the information they monitor, IDSs can be categorized into two main different types and related versions: host-based and network-based.

3.1.2.1. Host-based IDSs

During the premises of the intrusion detection field (in the early 1980's), IDSs were designed as *host-based* systems [Lai00]. They were focused on analyzing activities inside hosts only, network activity was not monitored. Networks did not have the importance they have today so they were not a major concern. A host-based IDS operates on the host its monitoring. It examines operating system's (OS) log files, accounting information of proceses, user behavior, or outputs from application level IDSs in a rather automated and real-time manner. Some host-based IDSs support a basic network-type detection by listening to port activity.

The host-based IDS family can be further categorized. The *application-based* IDSs focus on the application level. They examine the behavior of programs, generally analyzing log files. Some other IDSs are *stack-based* oriented; they watch packets as they go through the protocol layers. It allows pulling a dangerous packet from the stack before the OS or an application processes it.

3.1.2.2. Network-based IDSs

A *Network-based* IDS examines the network traffic, using raw packets as its data source [Lai00]. It captures and analyzes all the traffic in real-time. Usually, it includes a first level filter that selects the traffic to be passed on to an attack recognition module. It improves efficiency by filtering out known legitimate traffic. Filtering has to be configured with care as it must not discard any packets belonging to a potential attack.

An additional kind of IDS has been designed to operate at an upper level than the network-based IDS. It functions according to a *multi-network infrastructure* that correlates data coming from several networks inside a same administrative domain. The data are gathered and centralized so that a human team is able to watch over the whole administrative network and can carry out defensive actions when an intrusion happens. Every previously mentioned IDS type and even other multi-network systems are possible data sources of a multi-network IDS.

3.1.3. Current State of the Intrusion Detection Technology

The wide and diversified pool of current IDSs can roughly be divided into four main categories: research, commercial, public-domain or government off-the-shelf (GOTS) products [SPID pp. 17-45].

3.1.3.1. Research products

This first category is issued from early research efforts in intrusion detection back to the 1990's. Many IDSs were first developed by students to explore concepts and they were not maintained. However, these efforts have driven the orientation of the subsequent research and also led to the creation of commercial ventures. Primarily host-based oriented, the research focus changed towards network-based IDSs as the importance of networks increased quickly.

SRI (Stanford Research Institute) International's EMERALD is an example of a research tool. It was created to explore issues related to the use of both anomaly-based and misuse-based detection. The University of California at Santa Barbara has developed NetSTAT, from the STAT (State Transition Analysis Tool) line of research tools, which aims to use analysis of network-based systems' state-transitions to support real-time intrusion detection. The Lawrence Livermore National Laboratory's Bro partly focus on robustness of IDSs against attacks. It addresses also high-load monitoring, real-time notifications, decoupling mechanisms from security policy, and system extensibility.

3.1.3.2. Commercial products

Development of commercial products has been mainly focused on network-based IDSs but more recently it has made strong efforts to integrate both host-based and network-based approaches. Value of commercial products can hardly be pointed out as literature about them is directed towards marketing. Companies speak highly of their “comprehensive [...] security solution” (Cisco, on their web-page http://www.cisco.com/warp/public/cc/pd/sqsw/sqidsz/index.shtml), “comprehensive, end-to-end coverage of the attack timeline” (NFR Security on the front page of their website http://www.nfr.com/), “prevention and response for attacks and misuse” (Internet Security Systems (ISS) on their web-page http://www.iss.net/products_services/enterprise_protection/rsnetwork/sensor.php). Such claims tend to ensure customers that commercial products provide complete security mechanisms, which is wrong at present. This minimizes the importance of the necessary and permanent attention buyers have to spend on their products to maintain their performance. Thus, it cannot only threaten the security of the product users themselves, but also any other third party that can be targeted by an attack that is carried out using maliciously some vulnerable hosts. Moreover, as publicly available documentation is mostly made for marketing purposes, the real functionalities and important characteristics of the IDSs (such as false positive and false negative statistics) can be really hard to identify.

Many companies are competing today in the intrusion detection technology field. The tools named here only represent a few examples amongst the numerous existing commercial products. Cisco and ISS provide a wide range of IDS products, addressing both host-based and network-based aspects, for normal and high-speed networks. Latest Cisco products include the IDS 4200 series (network-based) and IDS Host Sensor (host-based). ISS has developed the RealSecure range of products. NFR Security commercializes the NFR Network Intrusion Systems. NFR is also an example of the immaturity of the commercial IDS business. NFR has recently bought Cybersafe, which was providing on a tool called Centrax. Centrax was before created under the name of Entrax by Centrax, a company Cybersafe once bought. This illustrates that aggressive competition still rules in the unstable IDS market.

3.1.3.3. Public-domain products

Public-domain IDSs have very varied origins. Certain public-domain tools were originally commercial tools that have been publicly released for various reasons (*loss leader* for another program, stopping the investments in development because of a lack of profitability, etc.). A loss leader is a product that is freely distributed or sold with a substantial discount in order to generate additional sales or to support the sales of another related product. Other products, like Tripwire, are both available in commercial and public-domain version. Tripwire is a file integrity assessment tool, first developed at Purdue University and now maintained in an open source project at http://www.tripwire.org. The specificity of Tripwire is that it aims to detect changes in the file system of the monitored system. Some tools, like Shadow, were supported by state organizations. Shadow architecture uses many sensors spread in important places on the network (including outside firewalls, for example) that forward periodically their data to a centralized analyzer. Finally, some projects have been entirely developed by the open source community, amongst which Snort is a well known example. Snort (http://www.snort.org) is a network-based IDS that can work in three different modes: straight packet sniffer, packet logger, or full blown network intrusion detection system. It works in real-time and its detection is misuse-based. It performs protocol analysis, content searching/matching and sends real-time alerts.

3.1.3.4. GOTS products

Government off-the-shelf IDSs are developed for government agencies that totally control the project. However, the work is not necessarily carried out by the agency's technical staff; it can be done by an external entity. GOTS products aim to protect networks like every other IDS but they differ in scope and focus. The USA's GOTS projects aim to protect national security. They also address four issues that commercial products do not: improvement of attacks from well-funded, nation-state attackers, intent identification, and objective evaluations of ID products. On the opposite, commercial products are designed according to totally different concerns: companies look for profits; they pursue the best practices because of their responsibilities regarding their customers.

CIDDS is a product developed by the USA Air Force Information Warfare Center. It has been designed as a fully dedicated hardware/software/operating system platform. It receives near real-time information and associated transcripts from its sensors. These data are stored in a local database that is then used for detailed correlation and analysis. Humans as well as automated tools can carry out these last operations.

3.1.4. Critics of IDSs

Both main types of IDSs present particular strengths over one another [Lai00]. This section first overviews these strengths and then present issues one faces when using IDSs.

3.1.4.1. Strengths of host-based IDSs

Host-based IDSs are less susceptible to *false positive* problems than network-based IDSs [Lai00]. A false positive is a misbehavior of an IDS that detects an attack while none has occurred. The opposite, when an IDS does not detect an attack that really takes place, is called *false negative.* Correct diagnostics are referenced as *true positive* and *true negative*. While normal traffic can sometimes look very close to attack traffic for a network-based IDS, host-based IDSs use logs of events that really occur. It enables them to know if attacks are successful or not so they are not likely to produce false alerts. Host-based IDSs also watch over specific system activities, with a particular look on key components. For example, they monitor events like administrator logins/logouts, access to specific files (password file, key executables), changes in access rights, etc. Network-based IDS cannot be aware of that kind of activities. Moreover, if a communication is encrypted at the application level, a network based IDS will not be able to analyze the real content of packets while the action triggered by the packets will be logged and analyzed by the host-based IDS. Host-based IDSs are not currently real-time systems, but the latest development trends push on that direction. Certain IDS have been designed to work on event-triggered mode; they do not rely on a process that periodically checks the content of log files but they react as soon as an event is logged. The last advantage of installing a host-based IDS is that it does not require additional hardware and it can be easily implemented on the existing network resources.

3.1.4.2. Strengths of network-based IDSs

One network-based IDS can be used to defend numerous devices. It has only to be placed at a critical access point in the network where it can watch over all the communications implying the systems it defends. Management costs are therefore reduced. Network-based IDSs detect malicious intents independently from their results. They examine all packet headers and/or payload and look for patterns that reflect potential attacks. On the contrary, host-based IDSs cannot detect unsuccessful attempts that failed to trigger any abnormal activity. Capturing packets with a network-based IDS enables to reliably gather evidence while attackers can elude host-based IDS detection by removing evidence of their actions. Usually, experienced crackers remove or damage audit log files. To remove the evidences gathered by a network-based IDS, a cracker would have to direct his attacks against it. However, network-based IDS do not need any actively communicating interface, they just have to silently record traffic. Therefore a network-based IDS does not need an IP address on the network it watches over, and other network components do not need knowledge of its presence. In that way network IDSs can be partially hidden and attackers are not able to detect nor to reach them directly. Network based IDSs can react quicker than host-based IDSs. Firstly, unlike host-based IDSs, network-based IDSs truly work in real-time: packets are analyzed as soon as they are captured. Secondly, as they intercept attack messages before all of them reach their target, IDSs can react a bit in advance, before attacks concretely takes place. On the contrary, host-based IDSs are more likely to react a bit after the attack was performed. Finally, data sources of network-based IDSs are not OS dependent while host-based IDSs rely on OS specific sources. For example, key executables and files are not the same on UNIX platforms as on Windows platforms. On the opposite, network protocols such as IP, TCP, UDP, are common standards in both environments. This has important implications on software development. It tends also to increase costs in multi-OS environments as each different OS requires a different IDS.

3.1.4.3. Issues about IDSs

To add an effective intrusion detection functionality in a network, there is clearly a need for both host-based and network-based IDSs. Both have strengths that the other one does not have and limitations that the other one overtakes. However, this complementarity cannot cover all the issues that IDSs face [Pta98].

Both misuse detection and anomaly detection methods have shortcomings. Misuse detection presuppose that the attack is known and identified. This means that the system is defenseless when a new kind of attack occurs. Attacks may also be difficult to identify when using patterns very close to the ones of legitimate operations. On the contrary, anomaly detection is not very flexible: every modification of the normal behavior of the system (adding a new service, for example) has to be reflected in the configuration of the IDS. Otherwise, the change will be interpreted as an attack. The IDS is also likely to produce a high amount of false positives as it can be very difficult to make an exhaustive pattern of the normal behavior of the system. Precision to distinguish attacks from normal behaviors is a major issue in that case.

Indeed, the effectiveness of IDSs is limited by the optimization of their configuration. When installing a new IDS, the system administrator needs time to configure it properly before it functions effectively. Moreover, this optimization process never ends as the configuration will have to take into account each newly discovered attack, each change in the services offered by the network, etc. However, having a good configuration is crucial as it sets the ability of the IDS to discover attacks. The IDS has to be able to detect any attack in a way as precise as possible. Too precise, a rule may not allow detecting all the variants of an attack. Too vague, a rule may include legitimate traffic in its attack definition. According to its level of optimization, an IDS will produce a more or less high amount of mis-identifications, but in any case perfect detection is impossible.

IDSs can also have security vulnerabilities like any other program. Even if the IDS is hidden (it does not have a valid IP address on the network it defends), attackers may exploit bugs. For example, on 2003 March 3, it has been discovered that the open-source IDS Snort had a buffer overflow vulnerability that could be remotely exploited [Dow03]. Snort could be led to run malicious code embedded in sniffed packets with super-user privileges. Thus, to disable Snort, an attacker did not need to obtain its network IP address and attack it directly.

The proper functioning of an IDS also depends highly on physical resources. Depending both on network's bandwidth and on its design, a network-based IDS may not be able to capture every packet that goes through an high speed network. A lack of CPU resources (that can be caused by a DoS attack) can lead an IDS to slow down its analysis. In that case, an IDS is not able to instantaneously detect or respond to an attack.

IDSs that are able to trigger reactive actions (for example blocking attack traffic at firewall) can be used as DoS tools by attackers. If a cracker is able to exploit false positives, he/she can abuse an IDS and makes it react to attacks that do not actually exist. Triggering blocking, the IDS denies service to some legitimate traffic.

Finally, network IDSs are vulnerable to subtle attacks that exploit the fact that the targeted end-system and the IDS may not analyze packets in the same way [Pta98]. These attacks use message fragmentation characteristics (in IP, for example) to achieve their goal: make the attack transparent to network-based IDSs. The first one, called *insertion*, uses packets that are accepted by the IDS and rejected by the target (figure 2). The attacker fragments its attack in *n* packets and inserts *m* other packets in it before sending it on the wire. Upon reception of the fragmented attack data stream, the network-based IDS reassembles the *n+m* packets that do not mean anything together and so the IDS does not react. On the contrary, the target does not accept the *m* packets and therefore only reassembles the *n* packets that contain the original attack.

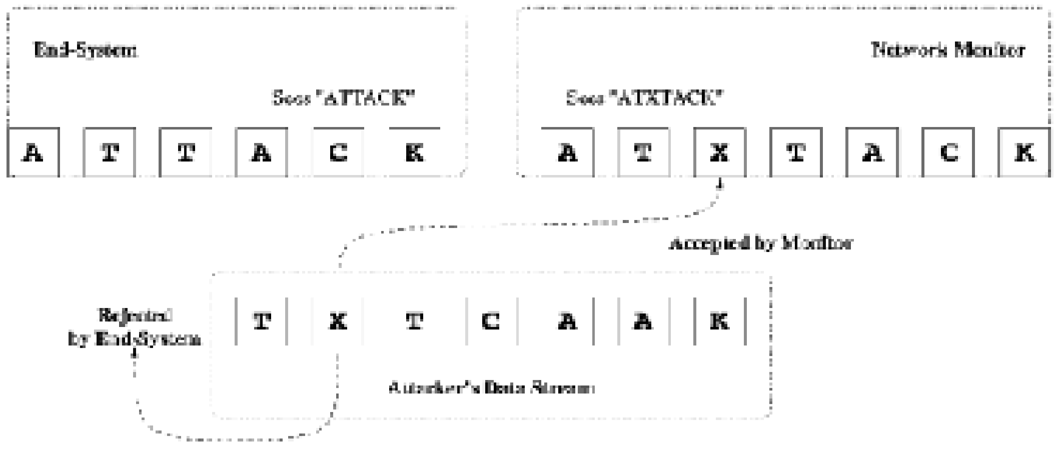

Figure 2: Insertion of the letter 'X' **[Pta98]**

The second attack, called *evasion*, exploits the opposite behavior: packets that are accepted by the target and rejected by the network-based IDS (figure 3). By conveying – partly or totally – its attack in such packets, the attacker makes the network-based IDS drop the whole or a part of the attack. Anyway, data gathered by the IDS do not mean anything. In the meanwhile, the target reassembles all the packets and the attack takes place.

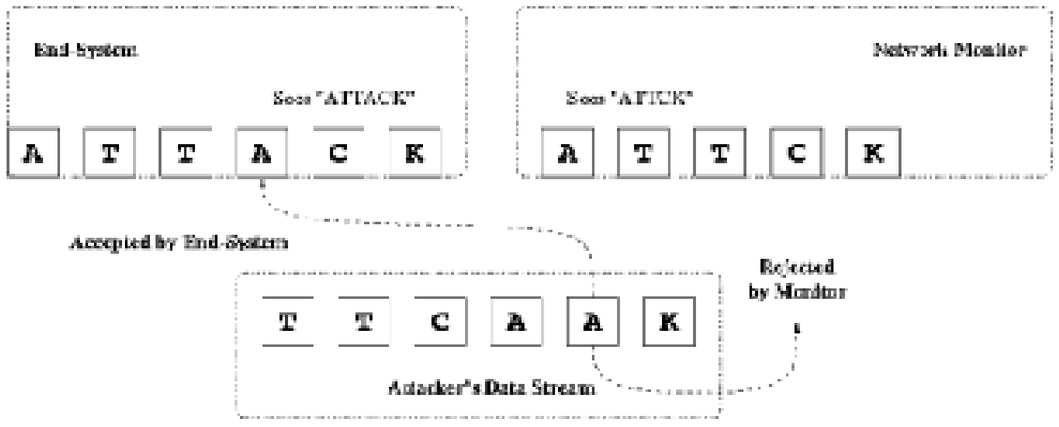

***Figure 3: Evasion of the letter 'A'* [Pta98]**

3.1.5. IDSs as Defense Means Against DoS Attacks

3.1.5.1. Framework

Since the first DoS attacks appeared, most of them have been “manually” detected. Anomaly-based IDSs may provide an alert when facing a new DoS attack. However, this requires a well-configured system, so that the attack does not fall in the “normal behavior” pattern, which is usually described in terms of statistical distributions or variances. Nevertheless, the probability for an IDS to detect a new attack is low, as shown in [Lip00]. Then, DoS attacks are mainly noticed when systems go down, when web-pages are defaced, when the Internet is not accessible anymore, etc. Once an attack has been noticed, administrators have to act urgently to identify its cause and find a response. The defense actions usually involve other parties, like the ISPs whose networks are used by the cracker (s) to propagate the attack. The process can last a few hours, and up to a few days. Then, the knowledge circulates so the computer security community is aware. Still at this point, the defense process is mainly manual.

Misuse-based IDSs can only provide automatic detection and trigger defensive actions after the initial identification of attacks. Once a DoS attack pattern is known, one can feed it in the signature database of a network-based IDS, thus enabling automatic detection. Consequently, IDSs' signature databases are regularly updated to face new attacks.

Security administrators have to be aware of the importance of maintaining up-to-date signature databases. Not doing so opens a breach in security policy, whose importance gradually increases with time.

When a network-based IDS detects a DoS attack, it can identify the harmful aggregate by extracting pieces of information from the attack packets such as source and destination IP addresses, port numbers, and so on. It can then trigger a defensive action based on these data. Until now, the automated reaction that has prevailed is to block the harmful traffic at network boundaries. The IDS sends a command to one or more routers and/or firewalls so they discard any packets belonging to the detected attack aggregate. In case human operation is needed, IDSs can alert administrators (for example via e-mail) as soon as they detect a DoS attack, providing an important gain in responsiveness.

3.1.5.2. Particular limitations of IDSs facing DoS attacks

As specified above, there is a potentially important gap of time between the first detection of a new DoS attack and the update of IDSs' signature database. Taking in account the time elapsed before security administrators effectively update their systems further increases that gap. In the meanwhile, the new DoS attack will be totally invisible to IDSs.

DoS attack are easily recognizable when they exploit a particular security vulnerability in their target. But a current trend in the cracker community is to make DoS attacks look more and more like legitimate network traffic. Thus, using anomaly-based detection, attacks may remain undetected because they will suit a normal behavior pattern. Using misuse-based detection, how can an IDS make the difference between the normal use of ICMP `Echo Request` by a program such as ping and an ICMP flooding attack? If the misuse detection relies on extensive patterns, it is likely to produce high amount of false positives, as explained above in the section 3.1.4.3.

Once again, maintaining up-to-date IDS signature databases is a critical issue. However, one can predict that numerous system administrators will not do it. Some are not aware of updates' importance, some others know but will not place it in their priorities. They can be overloaded by some other work and/or they do not perceive any threat since attacks – hopefully – seldom occur.

3.2. Quality of Service

The Quality of Service (QoS) field has not been designed as a security related concept. It aims to manage traffic aggregates in order to enhance the quality of network services. To achieve this goal, QoS technologies relies on traffic control mechanisms that can also be useful to mitigate DoS attacks.

3.2.1. Overview

A few years ago, the exploding development of the Internet posed some particular quality issues. The Internet was originally designed according to the *best effort* principle and every packet had the same access right to resources as any other. The best effort model is the simplest possible network service: it does not guarantee resource availability but assumes that applications are fault tolerant and include mechanisms (like retransmission of lost packets) that ensure reliability. This model was fine for the first Internet applications (e-mails, newsgroups, etc.) but it appeared that this model could not take into account the specificities and the diversity of many network applications' requirements that were developed later (video and audio stream applications, for example).

Quality of Service [Wan01] aims to solve that problem. QoS refers to the capability of a network to address two issues: *performance assurance* and *service differentiation.* Performance assurance refers to the need of ensuring end-to-end transmission reliability. In the Internet, the problem arises when resources run out. For example, when congestion happens in a network link, packets may be lost because they are dropped by overflowing buffers. The best effort model does not ensure that any packet will effectively reach its destination. It neither makes any difference between applications' various requirements or users' particular needs. Basically, the Internet can offer only one level of service while some think it would be better to offer different services according to needs; this is called service differentiation.

Nowadays, the approach that drives the development of the QoS field is to consider that most of the problems can be expressed in terms of *resource allocation*: when a network cannot meet the resource requirements of an application, packets get dropped or delayed. Work on that area aims to design mechanisms that actively control resource allocation so that every network application gets the right amount of resources it needs to work properly. To achieve these intentions, two technologies were developed: Integrated Services and Differentiated Services. The two next sections focus on these concepts as they include the useful components that are used later on in this thesis.

3.2.2. Integrated Services

3.2.2.1. Overview

The first attempts to bring quality support for real-time applications in the Internet led to the development of the Integrated Services (also called *IntServ*) [Bra94][Wan01, pp. 15-78] in the mid-1990's. Some researchers had demonstrated that *packet schedulers* could provide needed *delay* and bandwidth guarantees. Network delay is the length of time between the departure of a packet from a network traffic source and its arrival at destination. Packet schedulers are mechanisms used to decide which packets enqueued in a buffer of a network device should be first sent on the outgoing link.

The Integrated Service approach is based on an architecture using resource reservation on a per-flow basis. As Integrated Services have been designed to extend the QoS capabilities of the IP layer, they have to be supported by underlying layers to be effective. When an application wants to establish a communication using Integrated Services' capabilities with a remote peer, it has to set-up explicit resource reservation along the network path between them beforehand. The application first describes the characteristics of the flow and its resource requirements; this is called *flow specification.* Then, the flow should only be accepted if the network can meet these requirements. The path should be set according to resource availability, which should not necessarily be the shortest way. However, in current implementations, the default route is always chosen and the availability of resources is not checked. Still, Integrated Services particularly focus on providing good per-packet delay conditions because this is one of the most important issues for real-time applications.

Flow specification can be seen as a service contract between the source and the network. The source specifies the traffic and the network promises to commit resources according to it. If the source ever tries to use more resources than specified, the network will not guarantee any service for the excess traffic. Three parameters are commonly taken into account to describe flows' requirements: the *peak rate* (the highest rate at which a source can generate traffic), the *average rate* (the average sending rate over a time interval), and the *burst size* (the maximum amount of data that can be sent at peak rate). In Integrated Service, flows are often described according to the *token bucket* or the *leaky bucket* model (figure 4).

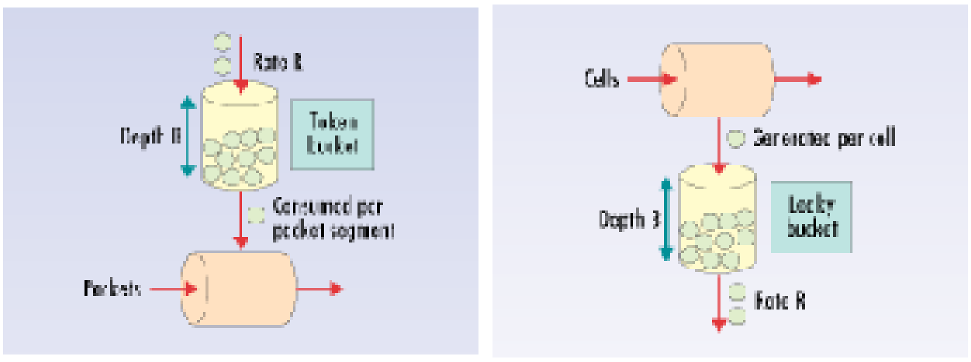

***Figure 4: The token bucket model (left) and the leaky bucket model (right)* [Chu02].**

A token bucket regulator consists of two parameters: the *token arrival rate* and the *bucket depth*. The token arrival rate is the rate at which *tokens* are accumulated into an imaginary bucket. Tokens correspond to a structure that contains a certain amount of bits. The bucket depth is the total amount of bits (i.e. total amount of tokens) that the regulator can store. When a flow goes through a token bucket regulator, each of its packets consumes a certain amount of the tokens stored in the bucket. The tokens are renewed at the speed of the token arrival rate and the total amount of stored tokens never exceed the bucket depth. If too few tokens are available, a packet is delayed until a sufficient number of tokens is present in the bucket. On the opposite, if the bucket is full, it allows a bursty flow to go through until all the tokens are consumed. To summarize, an application can approximately send packets through a token bucket regulator with a rate equal to the token arrival rate and a maximum burst size equal to the bucket depth. The leaky bucket model is slightly different from the token bucket regulator [Chu02]. Instead of accumulating tokens at a constant rate, packets are leaked from the bucket at a constant rate. This rate is characterized by a regular flow of tokens, thus, the packets in the bucket have to wait that enough tokens are available to be able to go through. This means that packet bursts are delayed during a certain length of time before leaving the bucket while in token bucket model the same time gap happens after the departure of the burst. Token buckets can be referred as credit-based schemes, while leaky buckets can be considered as deficit-based. As the figure 4 illustrates, in practice the token bucket model better suits traffic where packets have different sizes while the leaky bucket model is preferred to deal with fixed length data structures like the *cells* in the Asynchronous Transfer Mode (ATM) protocol. The cell is the basic data unit in ATM. Its length is fixed and comprises a 5 byte header and a 48 byte payload.

Service requirements are application-specific and are usually described by four required parameters: the minimum amount of bandwidth, the maximum delay, the maximum *jitter* (the difference between the smallest and largest delay experienced by packets from the same flow; however some different definitions are also used for this word), and the maximum *loss rate* (the ratio of lost packets compared to the total amount of packets transmitted). Each network node supporting Integrated Services knows a defined set of possible services and the corresponding parameter values. Two service models have been standardized within Integrated Services. *Guaranteed services* is meant for applications requiring the highest assurance on bandwidth and delay. It ensures a certain amount of bandwidth and a strict bound on end-to-end queuing delay. It takes into account the worst possible case when reserving resources. The *controlled load service* does not provide any quantitative or bound guarantees, but it tries to emulate a slightly loaded network. It allows *statistical multiplexing* and is implemented in a more efficient way than guaranteed services. Multiplexing is performed in link layer devices to arrange several different information streams (bit streams in data networks) into a common transmission medium (a link in fixed networks) that they all use at the same time to reach their destination. In statistical multiplexing, arrangement is performed according to the probability of the information streams' needs. The controlled load service fits well with adaptive applications that only need some degree of performance.

The Integrated Service model uses a particular protocol to set-up the resource reservation: the Resource ReSerVation Protocol (RSVP) [Bra97]. Hosts use it to specify service requirements to the network and routers use it to concretely reserve resources. RSVP establishes resource reservation in one way only, according to the receiver's requirements. In a two-way communication, both ends need to reserve resources. To do so, the sender first sends a `PATH` message towards the receiver(s). In this message, receivers find information about the traffic source, characteristics about the path and how to reach the senders. Then, they request resource reservation by sending back RESV messages along the exact reversed path than the PATH message. RESV messages set-up the reservation state in routers according to the receivers' specifications. Resources are not permanently reserved: reservations have to be refreshed. A timer in each network node along the path is set and resources are released if it expires. RSVP was designed to be independent from the underlying routing protocol and from the resource reservation policy: it only carries parameters that the relevant control modules process. For a last characteristic, “a reservation request can include a set of options collectively called *reservation styles*. Reservation styles determine how multiple requests are merged and which resource requests are forwarded to the upstream node” [Wan01, p. 44].

3.2.2.2. Key components of the Integrated Service architecture

Integrated Services rely on a reference model [Wan01, pp. 15-78] (schematized in the figure 5 hereafter), which is divided into a *control plane* and a *data plane*. The control plane is in charge of resource reservation while the data plane takes care of forwarding packets according to the reservation state.

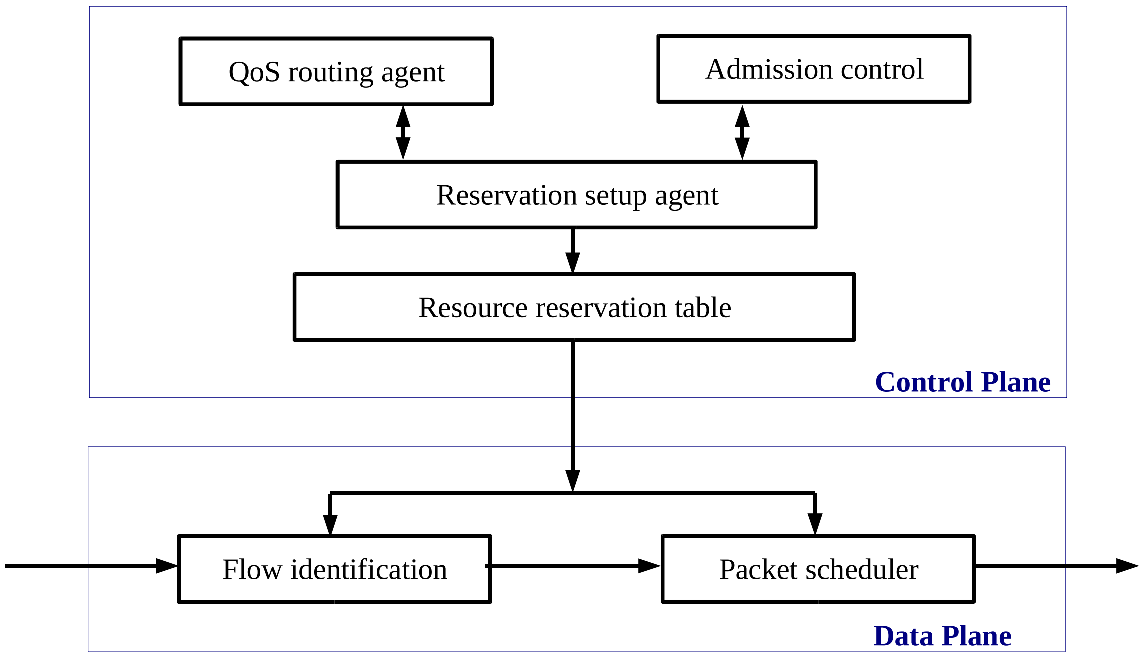

***Figure 5: Integrated Service reference model* [Wan01, p. 23]**

The control plane contains four elements: a *QoS routing agent*, an *admission control* module, a *reservation set-up agent*, and a *resource reservation table*. At each node, the QoS routing agent should be responsible for determining the path that should be used to set-up resource reservation. The selected path should be likely to have sufficient resources according to the requirements of the application that requests QoS. However, this is not the case currently. Route selection is not possible and only the default path is available, without any regard to resource availability. Routing and resource reservation modules are deliberately separated in this architecture as routing for supporting resource reservation is still a great problem. The resource reservation agent practically sets the reservation state in the considered node. For the Internet, it uses RSVP to set-up reservation hop by hop along the path. The resource reservation table is used to record the information about the reserved network resources. The admission control module takes care of guaranteeing resources for the reserved flows. Therefore, it denies reservation requests when there is not enough available resources in the node. Admission control can be *parameter based* or *measurement based*, the latter providing less guarantees on resource commitment. In the first approach, the traffic flow is precisely described by a set of parameters on which the admission control relies. The second approach is a probabilistic method that does not consider these parameters but measures the actual traffic load instead.

The data plane includes two particularly interesting mechanisms. The first element is the *flow identification* module, which aims to identify incoming packets and check if they match with any of the reserved RSVP flows. From each packet, the module extracts the *five-tuple*, which is a set of five elements: the IP source address, the IP destination address, the source port address, the destination port address and the transport protocol number. The five-tuple is then compared to the data stored in the resource reservation table. A match is found when the packet belongs to a RSVP reserved flow. Then, the *packet scheduler* enforces resource allocation. It directly affects flows' performances (particularly delays and bandwidth allocation) by selecting which packets get resources first. Its use is critical to ensure good service conditions, at least to the most important packets, when network resources diminish and queues build up in routers. Packet schedulers also have to optimize link usage by choosing the best way between *sharing* and *isolation* in a logical fashion. Sharing happens in datagram networks like the Internet where all packets get access to the same resources. The resource utilization is maximized but traffic flows disturb each others. Isolation is found in circuit-switched systems like in fixed telephony networks. In that case, each flow has dedicated resources, which are wasted when connections are not fully used. Different packet scheduling algorithms exist. Their designs differ according to characteristics like *work-conserving* or *non-work-conserving.* A scheduling algorithm is called work conserving if it is only idle when no packets wait to be transmitted. Most of the scheduling algorithms are work-conserving, but some propositions to reduce jitter are non-work-conserving algorithms. Schedulers can also be *simple priority* systems. In such systems, packets are assigned a certain priority level. Packets with higher precedence are always served first. However, these mechanisms only characterize scheduling algorithms, they do not categorize them. The three basic categories are *deadline based, rate based,* and *fair queuing.* Deadline based algorithms rely on the *earlier deadline first* (EDF) principle. According to this principle, every packet is assigned a deadline and packets are transmitted in the order of the deadlines. It allows decoupling delays and bandwidth at the cost of a complex admission control process. Rate based algorithms rely on two components: a regulator and the scheduler itself. The regulator shapes the traffic by determining the time at which a certain packet may be transmitted. Once transmittable, packets may be selected by the scheduler. This architecture enables the use of various regulators such as token bucket or peak rate regulators. The fair queuing category is the basis of Integrated Services. It allocates *weights* (a real number) to flows according to their importance. The bandwidth is proportionally shared according to flows' weights and unused bandwidth is proportionally allocated to backlogged flows, still based on their weights. The *Weighted Fair Queuing* is a widely implemented class of fair queuing algorithms that support bandwidth allocation and delay bounds. It is based on calculating a *finish time* (a number representing the order of the packets in the system) for each packet and it uses these data to schedule packets. Over the years, numerous variations of that algorithm have been developed.

3.2.3. Differentiated Services

3.2.3.1. Overview

In Differentiated Services (also called *DiffServ*) [Wan01, pp. 79-133] the traffic is divided in *forwarding classes,* which are groups that have the same predefined forwarding properties in terms of drop priority and bandwidth allocation. Forwarding classes receive different priority values and resource allocation is performed by provisioning resources according to classes' priorities. However, there is no absolute guarantee that the traffic gets the resources it requested. In the Differentiated Services framework, only nodes at the boundary of networks set the traffic's forwarding characteristics. Nodes inside networks only forward packets based on the specified forwarding classes.

Traffic behavior is not an end-to-end matter, a *forwarding treatment* is defined at each node. Forwarding treatment “refers to the externally observable behavior of a specific algorithm or mechanism that is implemented in a node”[Wan01, p. 83]. At a single node level, externally observable forwarding treatments are called *per-hop behaviors* (PHBs). The *Differentiated Service Field* [Nic98], redefines the IP packet header Type of Service (TOS) field so that it contains a six bit value (the *Differentiated Services Codepoint, DSCP*) used to encode packets' PHB. Two *PHB groups* have been standardized by the IETF: *Assured Forwarding* (AF) [Hei99] and *Expedited Forwarding* (EF) [Jac99]. A PHB group is a set of PHBs sharing a common constraint like the drop priority or the bandwidth allocation. Assured Forwarding is used by applications that require a better reliability than the one provided by the traditional best-effort. It defines four forwarding classes, to which are allocated a minimum amount of buffer capacity and bandwidth, and three *drop precedences* within each of them. Drop precedences are degrees of conformity of some network traffic with the resources it is offered on a certain network. When a traffic rate exceeds the bandwidth allocation for its forwarding class, packets with the highest drop precedence (the less in accordance with the guaranteed amount of resources) are discarded first. The framework requires that each Differentiated Service node implements the four AF classes and two out of three drop priorities. It is also required that packets within a same AF class are not reordered. Expedited Forwarding aims to “build a low loss, low latency, low jitter, assured bandwidth, end-to-end service through DS domains” [Jac99]. It relies on the principle that a traffic aggregate arriving at a node must leave it at a configurable minimum rate, independently of the intensity of any other traffic attempting to go through the node.

In QoS, *service* defines the overall performance that traffic experiences through a DS domain or end-to-end. While Integrated Services compulsorily need an end-to-end resource reservation, Differentiated Services can be defined for a single administrative domain. Deployment of services can be incrementally extended as providers agree on traffic characteristics' definitions. Details of the services are described in *Service Level Agreements* (SLAs), which are long term contracts between customers and their service providers. Characteristics such as traffic profiles (like flow specification in Integrated Services), performance metrics (throughput, delay, drop priorities), actions for non-in accordance packets, or additional marking and shaping services are generally considered. In general, only the traffic coming from customers into their providers' networks is liable to a SLA.

3.2.3.2. Key components of the Differentiated Service architecture

In the core of a Differentiated Service network, nodes only forward packets based on the configured forwarding treatments. The most important operations occur at network boundaries where the core components of the Differentiated Service framework are implemented [Wan01, pp. 92-98, 106-112]. Figure 6 illustrates these mechanisms.

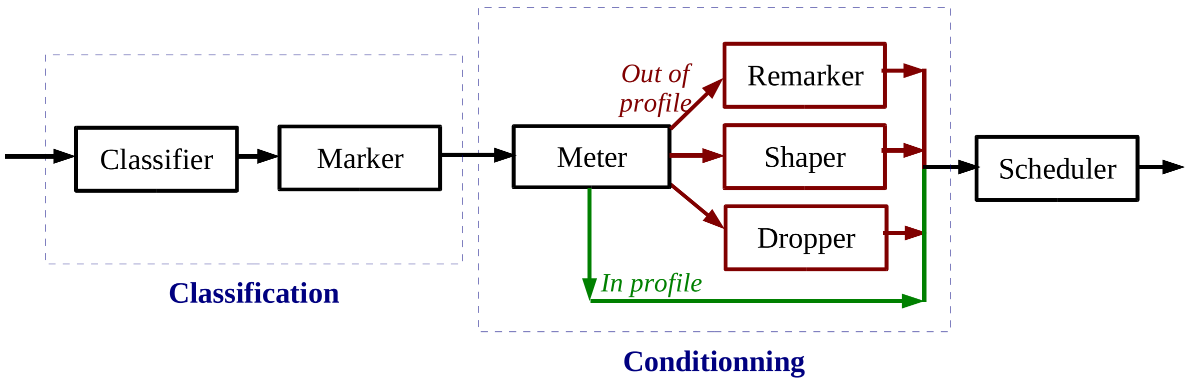

Figure 6: Classification, conditioning and scheduling **[Wan01, p. 93]**

Two main functions are performed in boundary nodes: *traffic classification* and *traffic conditioning*. The first one selects and marks packets according to the traffic profile they belong to. The second one enforces the *traffic policy* (set of rules governing the traffic's behavior): it measures the received traffic against its theoretical profile and forwards it or takes actions for out-of-profile packets.

“Packet classification is the process of identifying packets based on specified rules (also referred to as *packet filtering*)” [Wan01, p. 106]. The classification module is made up of a *classifier* and a *marker*. The classifier selects the incoming packet stream according to predefined rules defined by a *classification policy*. The classification policy may specify the way resources are allocated to traffic streams and the corresponding values of DSCP fields. Classifiers can be of two types: *behavior aggregate* or *multi-field*. Behavior aggregate classifiers base their decisions solely on the DSCP field. Therefore, this method works when the DSCP value has been set beforehand (e.g. in a prior Differentiated Service network). The multi-field classifier selects traffic according to one or more fields of the five-tuple in the IP packet headers. It can perform more complicated resource allocation (like application-specific allocation using port numbers). In the classification module, the marker simply sets the values of DSCP fields and adds packets to the right forwarding class.

In the traffic conditioning module, a *meter* first measures the traffic against its theoretical profile. Packets in accordance with the profile are allowed entering the network; others are further conditioned. Three different operations can then take place: out-of-profile packets can be remarked, shaped, or dropped. Packets can be remarked to indicate that they do not conform the traffic profile. Then, if congestion ever happens, these packets will be discarded first. Remarking is a frequent operation as packets go through many different networks that may have different traffic policies or that may use different DSCP values. *Shapers* delay packets so that the stream is slowed down to meet the profile's bounds. They are therefore used for stricter admission control, ensuring excessive packets do not enter into the network. *Droppers* can act much like shapers but using finite-size buffers so that packets are dropped when buffers overflow. They can also be configured to directly drop out-of-profile packets.

At the final stage, a packet scheduler (same component as in Integrated Services) is used so that traffic coming from every class can get access to an outgoing link. It reorders traffic from the different sources so it can fit into a unique common link. According to the rate characteristics of the classes, the outgoing link may be overloaded and in that case the scheduler discards packets.

Like in Integrated Services, traffic is often described according to the token bucket model. Therefore, metering and marking are often implemented by means of token bucket algorithms. Multiple versions can be used since Differentiated Services need to split traffic in more than two groups. Typically, Assured Forwarding can use a *dual token bucket filter* to implement the three drop priorities. Dual token bucket filters, such as the *two rate three* *color marker* (trTCM) [Hei99-2], divide streams in three different kinds of groups thanks to two successive token bucket regulators. The first one has a *peak information rate* (PIR) and a *peak burst size* (PBS) as parameters. The second one is configured using to parameters: the *committed information rate* (CIR) and the *committed burst size* (CBS) that have smaller values than PIR and PBS respectively. When making the contract, the service provider and its client agree on traffic characteristic. The service provider ensures the client will get a minimum bandwidth for its traffic in the provider's network. This guaranteed bandwidth corresponds to the committed information rate. A maximum sending rate is also agreed and it is represented by the peak information rate. The two burst size parameters enable some tolerance facing sudden traffic bursts that temporarily exceed the agreement's specifications. When using a dual token bucket filter, packets are allocated a color that represents the metering result (and which is often affected a certain drop priority). Packets in accordance with the parameters of both regulators are marked as green (the smallest drop priority). Packets are marked as yellow if they do pass the largest regulator but not the smallest one. Finally, if packets do not go through any of the regulators, they are marked as red (the highest drop priority). A *single rate three color marker* (srTCM) [Hei99-1] is a similar system, with the difference that output from the first regulator is directly fed on the input of the second one.

3.2.3.3. Random Early Detection

Differentiated Services also make use of end-system mechanisms to improve service quality. *Active Queue Management* (AQM) mechanisms make legitimate sources know about congestion their traffic flows suffer so that they can react accordingly (typically they reduce their sending rates). The *Random Early Detection* (RED) [Flo93] algorithm is one such mechanism, which is of particular interest later in this document. RED algorithms are implemented in routers to help manage buffer space in queues. It has been developed to be jointly used with TCP to warn about potential congestion in advance. RED principle is to randomly drop a few packets before a buffer gets really overloaded. When the TCP sources detect these losses, they can slow down their transmission rates (believing that the network is congested). Thus, real congestion can be avoided.

The RED algorithm works with two thresholds that represent a certain quantity of packets stored in the buffer. While the amount of enqueued packets does not exceed the first threshold, all the packets are forwarded. When the amount of packets excesses the first but not the second threshold, a small ratio of incoming packets is randomly dropped. The value of this drop probability linearly increases as the amount of stored packets increases too. The maximum probability is usually quite small, in the order of 2%. It comes into effect when the queue size reaches the second threshold. Finally, if the amount of stored packets exceeds the second threshold, every additional incoming packet is discarded. This behavior is summarized in the graph below:

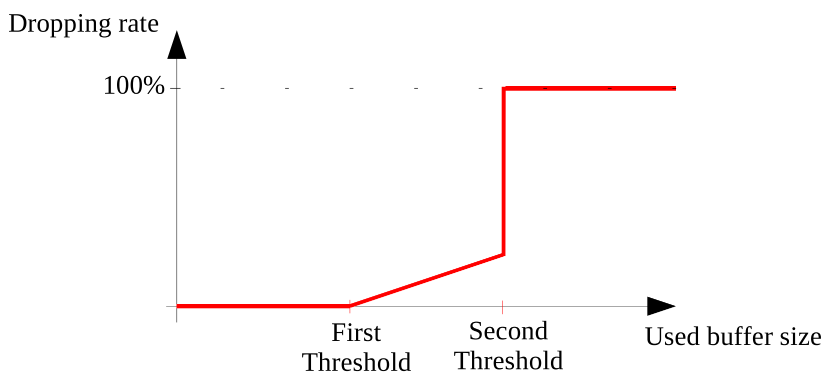

Figure 7: Dropping probability function in RED

4. MEANS OF DEFENSE AGAINST DoS ATTACKS

This chapter is meant to present the current state of the anti-DoS technologies. The task is hard for a simple reason: there are not any known means to protect a system against DoS attacks. However, preventive actions exist and some proposals have been published to address this problem. The end of this chapter presents the one on which the rest of this document is based.

4.1. Current Defense Methods

4.1.1. A Good Security Education

To defend against DoS attacks, there is no simple and comprehensive solution. However, DoS tools often rely on the exploit of known vulnerabilities. Their effectiveness is therefore often due to carelessness of security administrators who do not implement the known methods to fix security weaknesses in their systems. Every security administrator needs to be aware of the latest security trends and to follow the recommendations of security experts. This is an important requirement for the safety of the Internet community in general. Internet security involves everybody from users to service providers via software designers, and so on. Careless people do not only risk to be victims of crackers, they also threaten the systems or services of other Internet users.

The following practices should be followed by every Internet user or network administrator:

- keep informed. Organizations such as the Computer Emergency Response Team (CERT) regularly publish information about the latest security issues.
- apply security advice. When new problems arise, security organizations do provide advice to mitigate or fix them: these recommendations should always be followed.

- always install security patches or updates. DoS tools often use known software bugs. These bugs can often be fixed by applying a patch but effectiveness of DoS attacks prove that a lot of users do not do it.
- design a good security policy. A bad policy design leads to a bad effective security.

Moreover, companies should be concerned by hiring skillful security administrators as managing security in large networks requires a lot of care and competences. Software designers should always be aware of security issues in their programs. Around two third of the security advisories published by CERT in 2001 were related to buffer overflow bugs, while this problem and its solution are known since the 1980's.

4.1.2. DoS Related Defense Tools

A few security tools are currently available to help fighting against DoS attacks. Three of them are known to have useful capabilities: IDSs, firewalls, honeypots.

In that respect, the possibilities of IDSs were already addressed in detail. They can provide a very useful help by detecting known DoS attack signatures or abnormal behaviors.

At network boundaries, firewalls (or equivalent functionalities in routers) can be used to filter DoS traffic. The current practice when detecting a DoS attack is to discard the attack traffic. In certain cases, anti-DoS rules can be directly implemented in the firewall if the attack patterns are easily and reliably recognizable. Otherwise, firewalls can be used to automatically build a new blocking rule when another specialized component, such as an IDS, detects DoS traffic. Implementing *Network Address Translation* (NAT) in firewalls can be used to hide the real addresses of potential targets. At network interfaces, NAT changes IP addresses used in one network into other different IP addresses used in another network. Typically, when sending a packet to a wide public network like the Internet, the real addresses used by a host on a local private network can be mapped into a generic address that is valid in the wider network. When the same host receives a packet from the latter, the opposite process is performed. Then, outsiders see only one global address and cannot guess the local network addresses.

Honeypots can provide some degree of security against DDoS attacks too [McC03] [Wei02]. The main interest is to be able to study the attack and to learn about crackers' methods. Then, the real system can be protected according to the information that was gathered in the honeypot. The honeypot does not provide any good solution for the end target of a DoS attack, but it helps fighting against the building of DDoS networks. Indeed, network administrators can better defend their systems against any malicious uses of them. Thus, they are less likely to be used as agents during a DDoS attack. The honeypots can also track the crackers' actions so that these data can later be used in trials. Nevertheless, research is currently the main reason to use honeypots regarding DDoS attacks.

4.2. Latest Related Works

4.2.1. CITRA and IDIP

The CITRA (Cooperative Intrusion Traceback and Response Architecture) architecture [Sch01] aims to mitigate the effects of DoS attacks by using a rate-limiting mechanism, which is quite close to the system presented in the next chapter.

4.2.1.1. The CITRA architecture

The latest published version of CITRA uses a two-level organization. At the highest level, administrative domains controlled by a component named *Discovery Coordinator* (DC) are called *CITRA communities.* A DC is a device with human oversight that controls and monitors activity throughout a community. One community is then divided into *CITRA neighborhoods.* A neighborhood is a set of CITRA-enabled devices that are directly adjacent, i.e. that are not separated by any CITRA-enabled boundary controller such as routers or firewalls. Every CITRA-enabled device collects network audit data. If one of them detects an attack, it sends the attack identification data to its neighbors and it requests them to check whether they are or not on the attack path too. Neighbors compare the attack pattern with their own audited data and find out if they are on the attack path. If so, they repeat the request to their own neighbors. Thus, the attack is gradually traced back to its source or to the boundary of the CITRA system. In addition to tracing the attack, each CITRA-enabled device also performs an automated response defined according to a certain policy. Possible actions can be blocking the traffic or limiting its authorized bandwidth, for example.

4.2.1.2. The IDIP protocol

The CITRA architecture relies on a particular protocol named IDIP (Intrusion Detection and Isolation Protocol) [Sch00]. IDIP uses three major message types: `trace`, `report`, and `directive`. `trace` messages are sent when a CITRA-enabled device detects an attack. They include a description of the attack traffic pattern and a recommended action. According to these pieces of information, next CITRA-enabled devices can continue the attack tracing mechanism and they can decide to perform a response or not. `report` messages are copies of trace messages that are sent to the Discovery Coordinator. The DC can then reconstruct the attack path and optimize a global response. The DC sends `directive` messages to CITRA-enabled devices to set the global response. Directives can be `undo` messages to correct a previously taken action or `do` messages to add an action.

4.2.2. Aggregate-based Congestion Control (ACC) and Pushback Messages

The Aggregate-based Congestion Control [Mah01] uses a different approach of the problem, aimed against all kind of flooding problems. Flooding can be characterized by a certain aggregate pattern that matches traffic consuming a high part of the network bandwidth. In a first step, ACC aims to detect and rate-limit such aggregates. After that, it propagates rate-limiting along the reverse path followed by the high-bandwidth aggregate.

4.2.2.1. Local ACC principles

The *local ACC* mechanism [Mah01] carries out the detection and rate-limiting of the traffic causing congestion. The local ACC can be divided in three steps performed at the local network level. First, the system has to determine if it is seriously congested. When congestion occurs, the system aims to identify the aggregate using the biggest part of the bandwidth. In a third step, the *ACC agent* (a component installed in ACC capable routers) imposes bandwidth limitations on the aggregate during a certain length of time and possibly asks upstream systems to do likewise.

To detect congestion, the proposed mechanism periodically monitors packet drop rate in ACC agents' queues. Congestion happens when this rate exceeds a threshold defined by the system's policy. A small jitter is applied to the monitoring intervals so that it avoids synchronization effects and that an attacker could not predict the response pattern of ACC to a DoS attack. Giving the background to events could also help recognizing attacks similar to previously detected attempts.

Identification of harmful aggregates is a complex problem. The proposed method assumes, based on observations, that flooding attacks have either a common source or common destination prefix. Looking at the packet drop history, a list of high-bandwidth addresses (e.g. drop ratio exceeding twice the average) is created. As it has been observed that most of the websites' range of addresses can be contained in 24-bit prefixes, the high-bandwidth addresses are clustered into 24-bit prefixes. Then, for each cluster the systems tries to find a longer prefix that contains most of the drops. Finally, the clusters are sorted in decreasing order according to the number of drops they represent. Rate-limiting can now be imposed on the most important identified aggregates. If the system fails to find at least one narrowly defined aggregate, traffic is considered *undifferentiated* (i.e. not dominated by any particular aggregate) and no action is carried out.

Once the aggregates to be rate-limited are known, the ACC agent calculates their arrival rate and excess arrival rate (thanks to their packet drop rates and their sending rates at the output queue). Then, the minimum number of aggregates that could be rate-limited to sufficiently reduce the traffic is calculated, as well as the suitable rate-limiting values. According to these calculations, rate-limiting is applied and its relevance is periodically checked. When no longer needed, the ACC agent stops rate-limiting.

When an ACC agent detects an harmful aggregate, it can forward this information to upstream routers thanks to the Pushback mechanism. It enables them to take defensive measures too.

4.2.2.2. The Pushback mechanism

The Pushback mechanism aims to propagate rate-limiting to upstream routers [Mah01] thanks to the Pushback Message syntax [Flo01].

When the drop rate for a particular aggregate remains high for several seconds and/or when it has other information pointing out a DoS attack, an ACC agent invokes Pushback by calling the *Pushback agent* at the router. Pushback agents coordinates information from several ACC agents and uses Pushback messages to communicate with neighboring routers. The Pushback agent determines on which link(s) the harmful aggregates are coming from (*contributing links*) and sends a *Pushback* `request` *message* to the router at the other end of the link. Pushback `request` messages are used to advice upstream routers to rate-limit certain aggregates. They transport information such as data about aggregates, bandwidth limitation values, expiration times. When a router receives a Pushback `request` message, it can decide whether to rate-limit the specified aggregate or not, using local ACC mechanisms. It then decides if it further propagates the Pushback `request` message on the same basis than its predecessor used.

When a router has implemented traffic-limiting upon request from another router, it then sends back *Pushback* `status` *messages*. These messages report the arrival rate of the rate-limited aggregate at the sending router. They enable congested routers to decide of the usefulness of continuing the rate-limiting.

The last message defined is the *Pushback* `refresh` *message.* Rate-limiting is only effective during a fixed period of time. To make it lasts, routers send periodic Pushback `refresh` messages to their upstream peer. The structure of the `refresh` message is equivalent to the one of the `request` message.

4.2.3. Comments on CITRA and ACC regarding rate-limiting of DoS attacks.

4.2.3.1. Limitations of the test cases in CITRA

The CITRA architecture has been implemented and tested in laboratory. The article [Ste01] describes how experiments have been successfully performed and analyzes their results. However, as it appears from the paper, the tests only deal with well identified traffic aggregates. Only the attack traffic suffered rate-limiting while the legitimate traffic passed through the system without penalties.

However, perfect traffic aggregate identification is not currently possible. IDSs performances suffer from false positives. Anyway, if perfect attack detection was possible, why would one use rate-limiting while blocking would be more effective? The interest of rate-limiting is to avoid the damages caused by blocking on the legitimate traffic that is mis-detected as belonging to an attack. This aspect has not been taken into account in the tests of the [Ste01] study.

4.2.3.2. Limitations of the attack detection and identification in ACC

ACC has been designed to solve a congestion problem, it is not specifically meant to target DoS attacks. It cannot make any difference between a normal congestion due to a *flash crowd* (a sudden burst of legitimate traffic) and an abnormal congestion due to a DoS attack.

ACC is a router-driven approach. This means that detection and decisions are made at the router level. In the DoS or DDoS attack case, ACC does not provide any means to know the real effect of these attacks on their end target. While common misuse-based IDSs have some knowledge about the purpose of the packets they filter, ACC has not. ACC can only detect flooding attacks that consume all the bandwidth but it cannot detect low bandwidth attacks such as TCP `SYN` flooding. It has been calculated that 500 TCP `SYN` packets per second could be sufficient to perform an effective DoS attack against a single host [Dar00]. With a packet size of 64 bytes, it results in a rate of 256 kb/s: a small number compared to the available bandwidth in most corporations' networks. There is a high probability than this kind of attack does not saturate the network and, thus, remains undetected and successful.

ACC also relies on the identification of a congestion aggregate based on IP addresses. DoS attack patterns – and even more DDoS attack patterns – vary very much. Spoofed random IP addresses are commonly used in these attacks. In that case, ACC may probably not be able to identify a precise aggregate; the traffic appears undifferentiated. Then, no action is carried out and the attack is successful.

4.3. Suggesting a rate-limiting system to mitigate DoS attacks

4.3.1. Motivations

In [Ste01], the effects of rate-limiting on legitimate traffic have not been tested. However, as specified in [Möl-1], DoS attack detection is currently far from perfect and one can expect a high number of false positives in a real-life system. The utility of rate-limiting is to be found in sparing the legitimate traffic mis-detected as attack traffic. If detection was perfect or nearly perfect, blocking would be a better solution as it would not affect legitimate flows while discarding all the attack packets. This document aims to present an implementation of a system using rate-limiting as an automatic defense mechanism against DoS attacks, as described in [Möl-1]. This implementation was used to realize some tests whose goal was to point out the relevance and the suitability of using rate-limiting to mitigate DoS attacks. This study particularly focus on the effects of rate-limiting on legitimate traffic, as presented in [Möl-2].

4.3.2. Forces of the Problem

Using rate-limiting enables attack traffic to get in the defended system, which is undesirable. The smaller the value of rate-limiting is, the less legitimate traffic is damaged but also the more an attack is able to succeed. Implementing a very low rate-limiting is equivalent to leaving the system vulnerable. On the contrary, the higher the value of rate-limiting is, the more attack traffic but also legitimate traffic are discarded. Using strong rate-limiting, the legitimate service is disrupted as not enough packets survive. This is equivalent to blocking in a less effective way: some attack traffic can still get in the system. Is rate-limiting useful and/or usable then? This is clearly a question of trade-off between the effects of rate-limiting on attack and legitimate traffic.

4.3.3. Objectives

The next sentence summarizes the objectives that drove the work presented in this document:

“Implementing and testing an automated rate-limiting system to mitigate DoS attacks.”

A Rate-Limiting System (RLS) [Möl-1] is intended to be a network defense system designed to mitigate the effects of DoS attacks. It combines an attack detection system (like an IDS) with traffic control capabilities in routers (like the mechanisms used in QoS) to limit the rate of DoS attacks. Rate-limiting is preferred over blocking in an attempt to preserve legitimate traffic mis-identified as belonging to an attack. The RLS is also an automated system that can react quickly without any human intervention. Simultaneously, it is also aimed at providing *early warnings*, that is to say to notify that an attack is in progress while its effects are not yet perceptible [Man02]. The purposes of the RLS and a theoretical study about its effectiveness have been presented in [Möl-1].

The rest of this document first goes through the requirements to build the RLS. It then describes an implementation that was designed in order to test the viability of the rate-limiting mechanism. The end of the thesis presents the actual tests and their results that are the main objectives and interest of this study.

4.3.4. Scope

Rate-limiting does not suit every DoS attack type. Particularly, blocking can sometimes be a more effective mechanism: it depends on which method gives the best trade off between damages on legitimate traffic and mitigation of attack traffic. According to [Möl-1], rate-limiting is an effective mechanism if the DoS attack meet the following characteristics:

- the legitimate traffic must be packet loss tolerant. Rate-limiting is performed by discarding a certain proportion of incoming packets. Thus, it is not usable on packet loss intolerant traffic. In that case, as anyway the defense mechanism prevents legitimate applications to work properly, blocking should be preferred because it is more effective against the attack.

- “the attack bandwidth must be rather low”. A very high-bandwidth attack would imply to discard a very high proportion of packets. For any legitimate traffic suffering rate-limiting, this situation would be similar to blocking. The RLS is then useful with low bandwidth attacks such as some attacks using TCP SYN flooding, or to slow down an attack in its early phase while providing an early warning.

- “the probability for a DoS attack must be low”. As shown in [Möl-1], blocking is more effective when the DoS attack probability is high. When a DoS attack probability is high, it is most likely that most of the attacks detected are likely to be true positives. In that case, it costs less to block some legitimate traffic than to leave too much highly probable attack traffic penetrating into the network.

- the “attack should be non-destructive”. A *destructive DoS attack* destroys the target's ability to function, for example, by deleting or modifying important configuration files or power interruptions. This kind of attack does not rely on flooding techniques. Rate-limiting principle allows some attack packets to enter the protected network, which can be enough to perform a successful destructive attack.

- The false positive probability is too high to use blocking. When the system can reliably detect and precisely identify attack aggregates, blocking is a good solution since it does not harm legitimate traffic. However, the more the false positive probability increases, the less suitable blocking becomes as it damages legitimate traffic. On the contrary, rate-limiting is usable for much higher false positive probability than what blocking tolerates.

In addition to these characteristics, we realized an implementation that has its own limitations. The RLS was only designed for the purpose of a few tests. This has strongly affected the conception of the RLS components: they do not satisfy a real environment's requirements. Attention was mainly focused on the effects of packet discard ratios on the usability of different applications.

5. REQUIREMENTS TO BUILD A RATE-LIMITING SYSTEM AGAINST DoS ATTACKS

This section deals with the theoretical requirements that should be fulfilled when designing a Rate-Limiting System (RLS) to limit the effects of a DoS attack according to [Möl-1] and [Möl-2]. The general structure is also described but without addressing any practical issue. Technical details for an implementation in a Linux environment are discussed in the next chapter.

5.1. General Requirements

5.1.1. Overview

First of all, the environment in which the RLS is expected to function has to be described. The goal is to defend a computer network against a potential DoS attack. Therefore, this network has at least one interface with an non-secure network from where an attacker can operate using a DoS tool. As an example, the defended network can be a corporative network and the untrusted network can be the Internet. As the basic idea behind the RLS proposal is to jointly use an Intrusion Detection System and traffic control capabilities, the network has to include these services. The routers, in which the traffic control capabilities are implemented, can be located inside and at the edges of the defended network.

The RLS is intended to be an automatic response mechanism, reacting to potential attack detections. The goal is to replace human intervention that is too slow. Therefore, solely real-time pieces of software should be used. The automated deployment phase of a DDoS tool can take several hours (althought it has been claimed than the Internet could be infected in tens of seconds [Sta02]), but once the DDoS network is operational an attack almost instantaneously reaches its maximum effectiveness. Only a small delay can occur according to the synchronization quality of the DDoS tool or the clock accuracy if the attack is bound to a certain clock time. As an early-warning provider, the RLS has to include a mean (e-mail, phone message, etc.) to inform security administrators about any attack going on.

The RLS is not expected to replace human decisions. The system will particularly be sensitive to false positives as it directly relies on an IDS. IDSs easily generate false positives, especially when they are not properly configured. Due to the important impact the decisions of the RLS have on the network's performances, the system should include a monitoring module (like in CITRA [Sch01]) so that the administrator can thereafter control misbehaviors and fix them.

A final general requirement but not the less important is security. The RLS itself must not become the target of an attack. Therefore, every piece of software used has to be carefully configured and every security update has to be installed. The same care has to be applied when designing a new component of the system.

5.1.2. The RLS architecture

The RLS is a distributed system made up of four parts: information sources (such as IDSs), traffic control system (in routers), and two programs called the *RLS-controller* and the *RLS-agent* (figure 8).

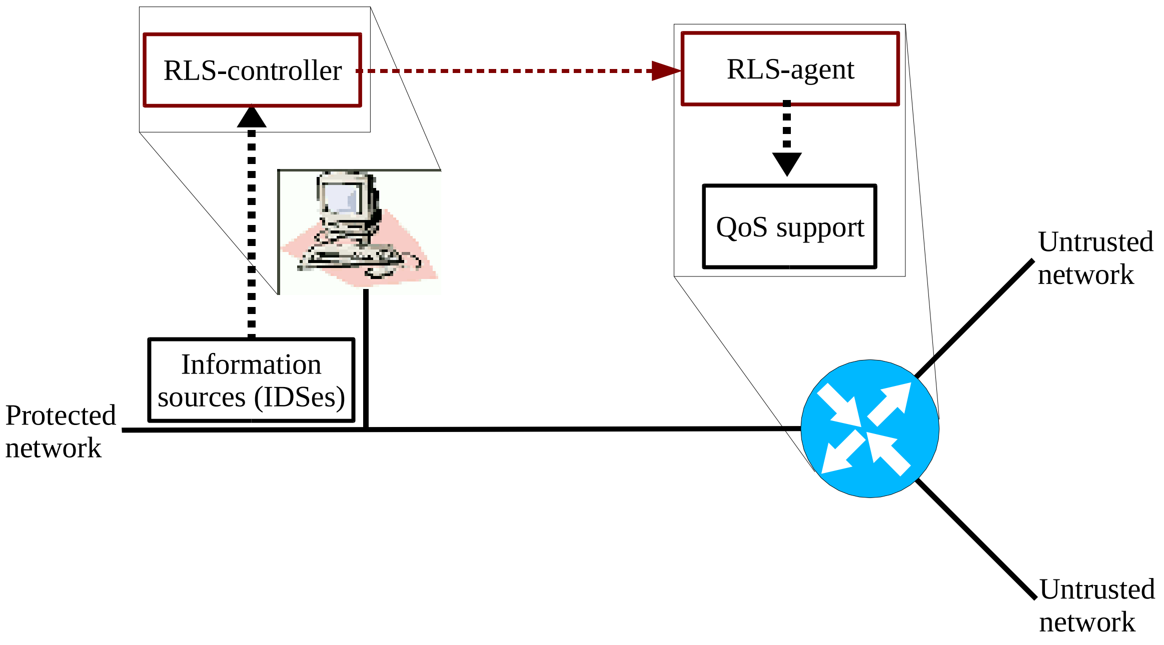

Figure 8: Generic architecture of the RLS.

The information sources take care about providing relevant input data for the RLS. In order to react to an attack and to rate-limit it, the RLS typically needs to know when a DoS attack is going on and what are the traffic flows involved. IDSs are a common tool for that purpose. It is worth noticing that the quality of the information stage will highly influence the reliability of the RLS. Therefore, combining a large number and a wide diversity of information sources can help to achieve better performances.

Data from the information stage are then fed into the RLS-controller. The generic architecture of the system is centralized around this component. The RLS-controller is intended to analyze data and it is responsible for controlling the RLS actions accordingly. It is particularly in charge of:

- the knowledge about the current state of the traffic control (which are the rate-limited flows, how long should they be rate-limited, what are their rate-limiting parameters, etc.)
- the identification of the attack aggregates. According to the data it receives, the RLS-controller has to identify the common pattern regrouping the attack flows in one or more aggregates.
- the rate-limiting decisions. According to the characteristics of the attack, the RLS-controller distributes to the RLS-agents a `RateLimit` message [Möl-2], which includes the identification of the aggregate and the rate-limiting parameters to be applied.
- the rate-limiting cancellations. Once rate-limiting is no more relevant for a particular aggregate (the attack has stopped), the RLS-controller cancels it.

The RLS-controller is also responsible for providing the early-warnings to the administrator. Also, as the RLS-controller is at the center of the system, it has to include the monitoring module.

The RLS-agents are located in traffic-control capable routers and they are used to enforce the rate-limiting of attack aggregates. Their actions are limited to applying the decisions from the RLS-controller. They build or cancel the relevant filters, set or delete the parameters of the traffic control components.

The traffic control part makes use of QoS mechanisms to implement the effective separation of the different traffic aggregates and to rate-limit the ones that are identified as belonging to an attack.

5.2. The Information Sources

The RLS rely on already existing tools to perform the attack detection. Such tools can be any system capable of detecting or giving indications of a DoS attack: IDSs from any kind, flow investigation mechanism such as the one used in ACC [Mah01], etc. The RLS should be independent from its data sources. It enables to use a good diversity and number of detection sources, which is a recommended method to implement an efficient defense mechanism. Using several IDSs of different types and located in different places in the defended network, a *correlator* can be used to combine data. “Alert correlation systems take as input the output produced by low level sensors such as intrusion detection systems, firewalls, and integrity checkers. Correlators issue reports that group together related alerts and events to provide an improved understanding of a cyber attack and to help analysts identify and dismiss false alarms.” [Hai03].

However, to keep the RLS-controller totally independent from its data sources, the alert messages should follow a standard, which does not exist. The design of the RLS-controller is therefore bound to be adapted to every data source used. Concerning their content, alert messages have at least to include the traffic flow identification data. The precision of the latter is only required to be as good as the precision required to differentiate traffic aggregates in the traffic control components (see the next subsection). Other information may also be useful, depending on the implementation of the RLS-controller.

5.3. The Traffic Control Stage

The RLS needs some components to enforce rate-limiting. Such components already exist in the DiffServ [Wan01, pp. 79-133] and IntServ [Wan01, pp. 15-78] architecture.

5.3.1. Using Components from the QoS Technology

As explained in [Möl-2], the rate-limiting mechanism should use the functionalities of QoS capable routers. Performing the rate-limiting in routers (whether than in end-hosts) aims to avoid overloading of end-hosts. It is also simpler and more cost efficient as no additional software is required in end-host. In a typical network, there are many more end-hosts than routers. Moreover, routers are a natural component of QoS architectures and a lot of them include QoS operations in their standard functionalities.

To differentiate network traffic between legitimate and attack traffic aggregates, several queues should be created and managed. The incoming traffic is first filtered so that traffic aggregates can be selected and sent to their right queue. This filtering operation, performed in accordance to a certain policy (here whether the traffic is considered legitimate or belonging to an attack), is called *packet classification*. Each queue is regulated by a particular Active Queue Management (AQM) mechanism, whose parameters determine the traffic conditions that a queue gets. Finally, a scheduler is needed to distribute packets on the outgoing link.

Packet classification faces a dilemma: the less complex it is, the faster it is but at the same time the less precise it is. Lack of precision increases the probability of including legitimate traffic in attack aggregates while the lack of speed damages QoS. Using the 5-tuples of the IP headers is assumed to be a good trade-off. 5-tuple filters can operate at high wire speeds [Lun03].

Rate-limiting can be enforced by the AQM mechanism or by the scheduler. The AQM mechanism appears to have more advantages as it can discard quite accurately a certain proportion of traffic without requiring information about the bandwidth consumed [Möl-2]. On the contrary, a scheduler needs a reliable estimate of these data while the fast variations of DoS attacks make it difficult. A scheduler would neither rate-limit an attack that would use less bandwidth than the configured bound. Therefore, the AQM mechanisms should be the actual traffic shapers and the scheduler has to be chosen and configured so it does not trigger an overflow in any queue. However, there is no special constraint for the scheduler as the RLS is meant to deal with low bandwidth attacks. The transmission capacity is not the bottleneck of the system. The size of the attack queue should be nevertheless large enough to prevent a possible small overflow. If the queue gets overflowed on the long term, it means that the attack rate is quite high and therefore it is no more in the scope of the RLS. In this case, another action, such as blocking, has to be done.

5.3.2. Design of the Rate-Limiting Functionality

Both IntServ and DiffServ capable routers can be used to implement a rate-limiting mechanism. However, several components are useless, particularly in the IntServ model. The DiffServ architecture is closer to the RLS requirements.

The RLS does not need to use the whole Integrated Service component architecture. For example, there is no interest in using a routing agent since operations are carried out at a single point. More globally, as represented in the figure 9 hereafter, using the control plane does not make sense. As the RLS-controller can provide the filtering rules (aggregate identification) and traffic policy (rate-limiting parameters), it replaces the whole control plane. An alternative solution could however make use of the resource reservation table by writing necessary information from the RLS-controller in it.

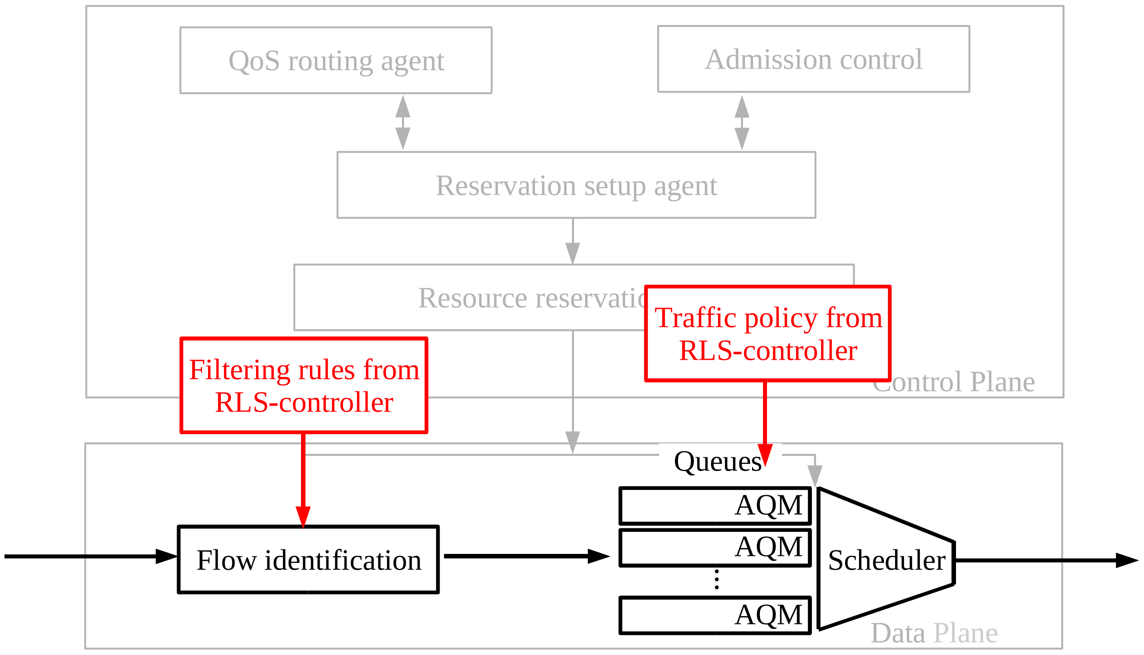

Figure 9: Traffic control architecture inspired from the Integrated Service model

Basing the traffic control structure on the Differentiated Service scheme does not require as many modifications as in the Integrated Service case. Basically, the only feature that is not necessary is the marking stage: rate-limiting is only implemented at a single network node. Therefore, its implementation is much more straightforward and easier than with the model derived from Integrated Services. The following figure represents how traffic control can be implemented using the Differentiated Service architecture.

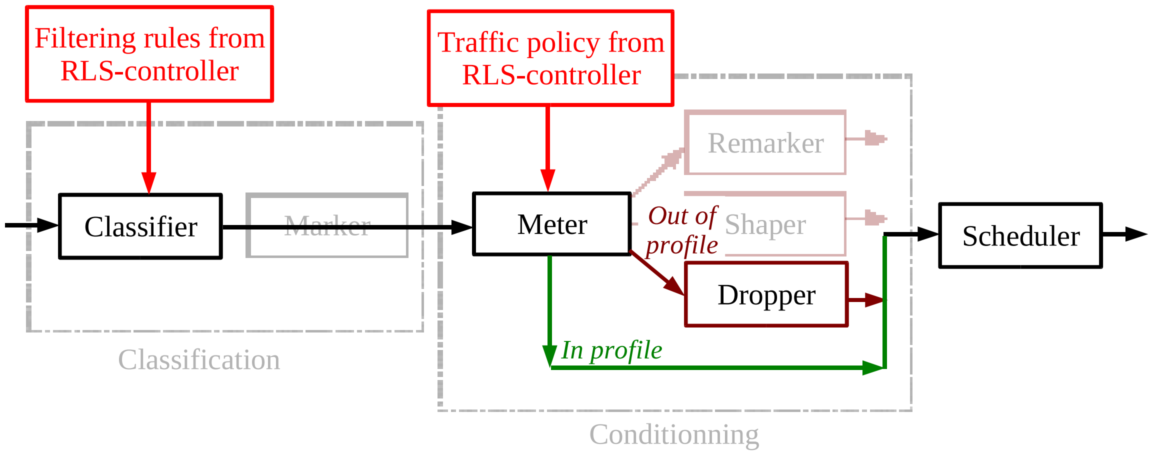

Figure 10: Traffic control architecture inspired from the Differentiated Service model

5.4. Keeping Knowledge of the Rate-Limiting State

The RLS distributed architecture raises a question about the memory state of the generic system. It is necessary that the system keeps information about the rate-limited traffic aggregates. Actions are not performed once and for all: traffic control in the routers has to be dynamic to follow the evolutions of DoS attacks. The system has also to remember the actions it did in order to avoid repeating them needlessly. For example, reception of an alert message from the IDS should not automatically lead to sending of a message to rate-limit the attack: the packet that triggered off the alert may be part of an aggregate that is already rate-limited since a certain proportion of the attack traffic does enter the network.

Basically, permanent knowledge of the traffic settings can be implemented in the RLS-controller, in the RLS-agents, or in both. The accepted proposition will affect the memory state of the modules: whether hard or soft. *Hard state* is a term used to define a system that keeps permanent memory of its working state. On the opposite, in *soft state* mode a system does not keep memory of any of its actions.

A first possibility would be to keep both RLS-controller and RLS-agent in a hard state. This redundancy enables a better resistance against crashes: every component is then able to retrieve its original state by itself. However, this method requires a mechanism to ensure that information in every component is coherent.

A second method is to set solely the RLS-agents in a hard state. This would mean that the RLS-controller would send alert notification messages and the RLS-agents would decide whether or not they should apply traffic-limiting. This solution has two drawbacks. First, it would generate a lot of useless messages as the RLS-controller would generate messages for each detected DoS packet: it would not know if it has already sent a similar message. Then, the administrator would not be able to modify settings from the RLS-controller.

The third way is to keep memory of the traffic control settings in the RLS-controller. This is the recommended centralized method. The RLS-controller can then check the relevance of the alerts from the NIDS before sending any message. Implementing a monitoring module so that administrators can perform manual actions is also made easier in that case. The possibility of crashes should be taken into account and the RLS-controller should include a mechanism to recover the correct system state by itself. Hard copies of traffic-limiting data with validity timestamps can be used for this purpose. If an RLS-agent crashes, it should alert the RLS-controller when starting up again. The RLS-controller would then send the right data for the RLS-agent to recover the right state.

5.5. Communications Between RLS Components

5.5.1. Communications over Four Interfaces

As the RLS is distributed, it requires a lot of communication means between its different components. The need for communication can be identified for four different interfaces in the system:

- between the information sources and the RLS-controller
- between the RLS-controller and the RLS-agent
- between the RLS-controller and the security administrator
- between the RLS-agent and the traffic control module

Out of these four interfaces, only two present a total liberty of communication design: the interface between the RLS-controller and the RLS-agent and the interface between the RLS-controller and the system administrator. Other interfaces are dependent one on the information sources, the other one on the traffic control program. The design of the RLS has to be adapted to the communication methods of these components. Modifying them should only be considered if they do not provide the real-time requirements and the exchange of all useful information required by the RLS.

Concerning the exchanges between the security administrator and the RLS-controller, there are no particular constraints. Alerts from the RLS-controller should be sent quickly and reliably, which can fit with many means such as e-mails, SMS, etc. The administrator's actions on the system should be operated via a human/machine interface and is only required to be as user-friendly as possible.

As to the exchange of messages between the RLS-controller and the RLS-agents, the design possibilities are also very open but they should be chosen with care as constraints are more important and effectiveness is more critical. Two issues have to be carefully considered: the distribution mechanism and the payload's content [Möl-2]. The optimal peer will be:

- reliable. If a rate-limiting command is lost, the defense mechanism does not work.
- real-time. The RLS has to be able to react as fast as possible when a DoS attack is detected and to be able to follow any quick change in the attack pattern.
- supporting multicast. In the general case, it is most probable that one RLS-controller will supervise several RLS-agents, sometimes repeating the same messages. Multicast is more convenient and efficient than unicast in that case.
- scalable. The design should be able to suit a minimum configuration with only one RLS-agent as well as a complex network architecture with several, potentially tens of routers.
- secure. This is a requirement for any communication protocol.
- the most efficient possible. Bandwidth is a precious enough resource to avoid wasting it.

5.5.2. Distribution Mechanism of the RLS-controller to RLS-agent Messages

Most of the operational requirements of the distribution mechanism rely on the transport protocol. Based on existing protocols, the distribution method can be one amongst point-to-point transmission (UDP, TCP), flooding (flooding protocol of OSPF, OFP), multicasting (UDP, MTP, RMTP), or implementing an extension to a routing protocol (OSPF, BGP) [Möl-2]. These choices differ in terms of transmission delay, induced extra bandwidth, scalability, etc. The most optimal choice is highly dependent on the network environment. If only one router is used, a point-to-point protocol can be suitable. If the network is connected to the Internet via several routers, multicasting can be preferable as messages will be the same for all the routers. The RLS should be able to operate efficiently in both cases.

The real-time requirement is above all a matter of transport protocol. Coupled together with an efficient usage of the bandwidth, protocols that use flooding and/or refreshment operations do not seem to be the most optimal choice. However, a protocol like UDP, which is often used to transport real-time applications, suits the real-time requirement but it does not achieve the reliability the RLS needs. In such a case, a mechanism to ensure reliability (using retransmissions/acknowledgments, for example) has to be implemented in an upper protocol layer.

From a security point of view, the design should particularly take into account authentication of peers (to avoid a malicious user to act as the RLS-controller and command rate-limiting of legitimate flows, for example), the reliability and availability of the system (an incoming DoS attack must not prevent the RLS-controller to communicate with the RLS-agents), the integrity of the messages (an attacker must not be able to modify the traffic flow identification), and the confidentiality of the messages (it helps to hide the system and to avoid an attacker gets any feedback on its actions against the system).

5.5.3. The Payload Content of the RLS-controller to RLS-agent Messages

The payload of the messages can have two functions. The obvious one is to carry the real data of the message. However the payload can also help to achieve needs that the transport protocol do not fulfill. For example, if the transport protocol lacks reliability, the messages can include a retransmission mechanism. Security methods (like checksum) can also be implemented in the payload.

Two RLS messages have been defined so far: `RateLimit` and `Cancel` [Möl-2]. The first one is used to request the rate-limiting of a new traffic aggregate. In its most basic form, the RateLimit message provides the aggregate identifications from which the RLS-agent builds filters to separate attack and legitimate traffic flows. A more fine-tuned message can include data to customize the traffic control parameters (rate-limiting value, AQM parameters, etc.). The `Cancel` message is used to remove rate-limiting on a particular aggregate. The only relevant information is the identification data of this aggregate.

Two known possibilities that fit the needs of RLS are the Pushback-messages [Flo01] and the messages defined in the IDIP [Sch00]. Both specify a message format for describing attacks and include rate-limiting information. These specifications allow additional message definitions so supplementary functionalities can be easily added.

6. SPECIFICATIONS FOR A LINUX IMPLEMENTATION

This section describes a simple RLS implementation in a Linux environment, as presented in [Möl-2]. This system has been designed for the purpose of a few tests, mainly to validate the possibility and usefulness of rate-limiting DoS attacks. As is, the implementation is not ready to operate in a real environment. Several specifications that were not affecting the test results have not been considered.

6.1. The Test Environment and the RLS Architecture

This section overviews the whole system. A description of the implementation and the sources and explanations to build a similar system are provided in this section.

6.1.1. Overview of the Test Network and the RLS

The test network (figure 11) is made up of three hosts running the Linux kernel 2.4.20. It is divided into two different, statically routed physical networks: Network 1 (10 Mb/s Ethernet) and Network 2 (100 Mb/s Ethernet). In the real life, these two networks would typically be an organization network and the public Internet.

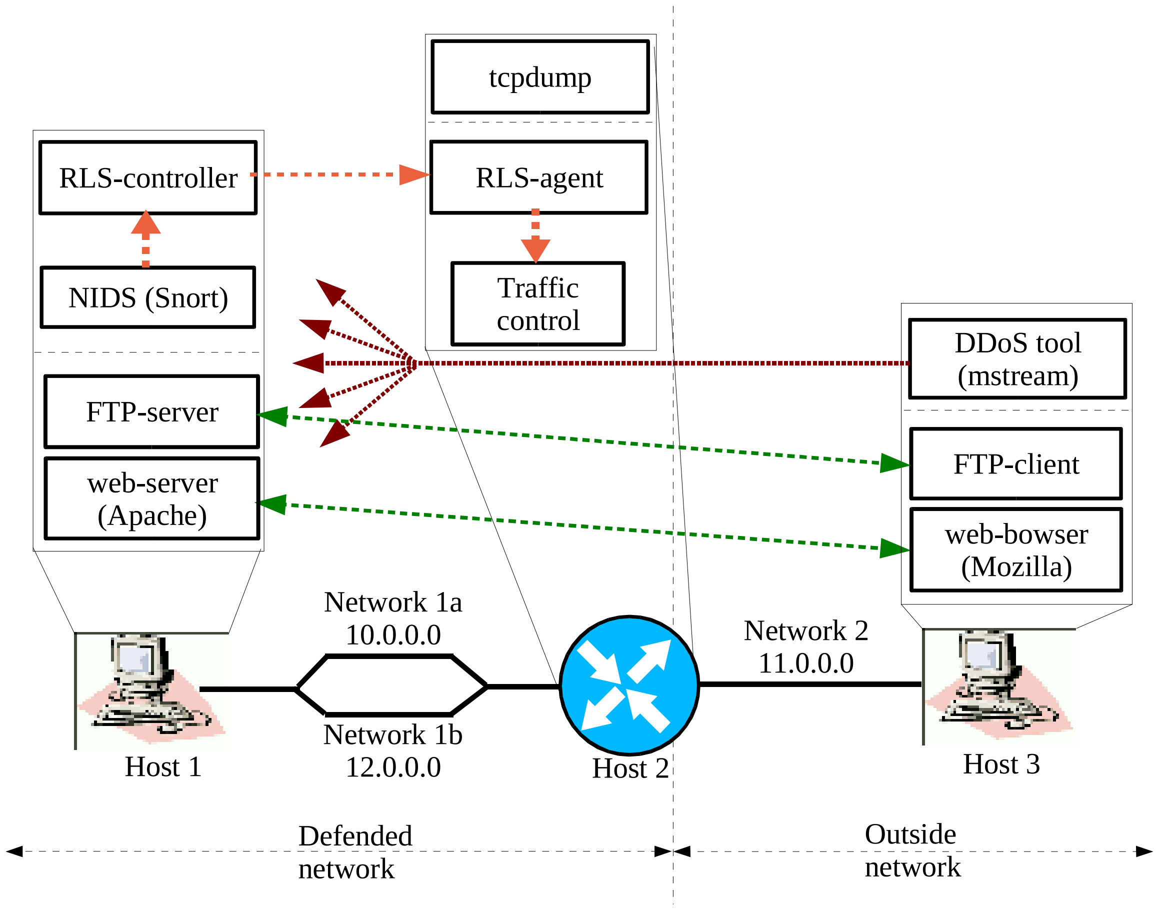

Figure 11: The test network and the RLS architecture

Network 1 is the network that is defended. It is linked to Network 2 via a router (Host 2). As the tests do not address the reliability of the attack detection, only IP addresses are used to differentiate traffic aggregates. Two traffic aggregates should be particularly differentiated: the traffic from legitimate users and the traffic identified as part of an attack, whether the attack detection is correct or not. In the actual network configuration, a problem arises as the same computers are used for generating both legitimate and attack traffic. To enable traffic differentiation, the Network 1 is split into two logical networks with their own address (Network 1a and Network 1b). Host 1 is then reachable via two different IP addresses, each of whose can be used by a different traffic aggregate.

Host 1 is a server providing two TCP-based services, HTTP and FTP, which can be used from the outside network. The HTTP service is assured by Apache, which is the most commonly used web-server on the Internet. Thus, Host 1 is also a typical target when carrying out DoS attacks from Host 3. The second function of Host 1 is to run two components of the RLS: the RLS-controller (described hereafter in section 6.2.) and an IDS. The IDS is Snort 2.0, which is an open source, misuse based NIDS. It is the only information source the test system uses. The NIDS watches over the traffic on Network 1a and Network 1b.

Host 2 acts as an upstream router. It includes traffic control capabilities that are provided by the Linux kernel. The TC utility is a command line interface to the traffic control functions of the kernel. This tool can be used to enforce the rate-limiting operations. The TC commands are executed by the RLS-agent according to the messages generated by the RLS-controller. TC command lines look like the following:

```
tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip src
#SRC match ip dst #DST flowid 1:1
```

In this example, TC is used to add a filter on the network interface named `eth0`, on which the queues are managed by a scheduler identified by `1:0`. This filter is given the priority 1; this number defines the order in which several different filters should be applied. This filter uses a u32 selector to select the packets; the u32 selector is used by TC to base traffic selection on any desired fields within the packets. In this example the filter matches every IP packet whose source address and destination address matches the macros `#SRC` and `#DST`, respectively. The matching packets are identified as belonging to the flow `1:1`, to which a particular queue is alloted and scheduled by `1:0`.

tcpdump is installed on Host 2 to record the necessary packet statistics to be analyzed. tcpdump is meant to watch over a network interface and to record information about the packets going through that interface. It can provide precise information such as the protocols used, the source and destination addresses, header options, contents of messages, and so on. tcpdump can record partial information from the headers or it can be also used to record entire packets. On Host 2, tcpdump is used to capture packets on both Network 1 and Network 2 interfaces.

Host 3 includes programs for both legitimate and malicious users. Legitimate traffic is generated from a simple FTP-client and a graphical interface web-browser (Mozilla). The ping command was also used during the tests. Attacks are performed using the mstream DDoS tool [CER00-2], whose attack pattern is included in Snort's signature database. mstream works according to the handler/agent architecture. The handler can request a TCP

`SYN` flooding attack against one or more IP addresses. Since its discovery, this tool has been largely studied and the source code can be found on the Internet. To be able to use it in the test environment, the source code had to be slightly modified. First, the handler and agent programs were changed so they can operate from the same computer instead of relying on a remote connection. To do so, messages are exchanged through a local Unix socket instead of using network sockets. Then, the agent's sending rate had to be slowed down. The original code sends attack packets as fast as the computer can. The router of the test network does not have the capability to process every packet in time under those conditions; packets are dropped. As all the packets have to be recorded to produce reliable packet statistics before analysis, changes in the code were required so that the mstream agent sends packets as fast as possible within the limits of the capacities of the router.

6.1.2. How to Implement the Already Existing Components

The test environment was set-up using freely available software, mainly from open source projects. This section explains how to gather the required programs and the related documentation. Some advice are also provided for specific configurations.

Common to the three stations is the operating system kernel: Linux. Several companies, such as Debian (http://www.debian.org) or Red Hat (http://www.redhat.com), offer free operating systems based on the Linux kernel. A lot of various software packages are often also provided. The operating systems are usually available on the companies' websites and distributions (including various packages) are also edited on CD-ROMs. Note that traffic control functionalities have been only added in the latest versions of the Linux kernel. The most recent kernel has also the most comprehensive set of QoS capabilities. It can be found on the website http://www.kernel.org. User manuals are also available from the websites quoted in this paragraph.

Host 1 uses more particularly three pieces of software: the Apache web-server, the Snort IDS, and a ftp server. The latest versions of the two first programs and documentation about them are freely available on the following websites: http://www.apache.org and http://www.snort.org. To implement the RLS, Snort has to be used in alert mode with the option `-A unsock` to send the alerts to the RLS-controller through a local socket. Snort requires the libpcap library (used to capture network packets) to function. The source is available on the tcpdump/libpcap website: http://www.tcpdump.org. Concerning the ftp server, different versions are basically included in every Linux distribution.

Host 2 also requires the libpcap library. It is needed to install the tcpdump utility that can be obtained from the same website. The tcpdump sources include man pages to explain how to use the program. The second piece of software to be installed on Host 2 is the TC utility. This program is part of the iproute2 package, which can be downloaded on several ftp mirrors on the Internet, amongst whose: ftp://ftp.sunet.se/pub/Linux/ip-routing/. The best related documentation is to be found on the http://www.lartc.org website, where [Hub03] can be downloaded. Finally, Host 2 is a router, so it has to be configured as such. Static routing is enough in the test network we described.

Concerning Host 3, a ftp client is included with the operating system. Several freely available web-browsers exist, such as Mozilla that can be downloaded on the http://www.mozilla.org website. The mstream DDoS tool is also available on the Internet, one can use search engines and links from security related websites to find it.

Installing the test environment requires some network configurations that can be made with Linux commands. To add IP addresses on a unique physical interface, one can use the `ifconfig` command (for example: `ifconfig eth0:1 12.12.12.10 netmask 255.0.0.0 broadcast 12.255.255.255`). Adding new static routes is easily done using the `route` command (for example: `route add -net 11.0.0.0 gw 10.10.10.11 netmask 255.0.0.0`). Enabling forwarding of IP packets requires to set the content of the /proc/sys/net/ ipv4/ip_forward file to 1.

6.2. Designing the Additional Components

Three required pieces of software do not exist and have to be designed. Two of them are the core components of the RLS: the RLS-controller, the RLS-agent. Here, the tasks they perform and how they communicate are explained in detail. The third program is a suitable AQM mechanism to carry out rate-limiting.

6.2.1. The RLS-controller

The RLS-controller has been designed as a user-space program written in C. In the implementation, its main tasks are to analyze alert messages from the NIDS and to send `RateLimit` messages to the RLS-agent. It also sends alerts to the system administrator via e-mail messages.

When the RLS-controller starts, it first creates a Unix socket through which Snort can send alerts. The RLS-controller has to be started first because the IDS connects itself to the local socket at start-up. Then, the RLS-controller listens for incoming messages.

When an attack occurs, Snort reacts if its pattern is included in its signature database. Typically, an attack signature in Snort resembles the following:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 15104 (msg:"DDOS mstream client
to handler"; flags: S; reference:arachnids,111; reference:cve,CAN-2000-
0138; classtype:attempted-dos; sid:249; rev:1;)
```

The first part (in red) describes the characteristics of the packets that correspond to the attack. Here, any IP packet using TCP as transport protocol, coming from any IP address on the external network and using any port number, and targeting any address on the home network on port number 15104, will be regarded as a message from a mstream DDoS tool agent to its handler (in blue). The data included between the parenthesis is the part of the alert that describes the event. When detecting an attack in alert mode, Snorts can send these data through a Unix socket along with the raw packet that triggered the alert.

The RLS-controller receives all the alerts from the NIDS. It then selects the relevant ones that describe a DoS attack. The green part in the sample signature above is the most useful information for this purpose: it represents the class of the attack. Each rule is alloted a certain `classtype` that characterizes the threat (attempted administrator privilege gain, Denial of Service, detection of a network scan, etc.) and that is represented by a keyword (attempted-admin, successful-dos, network-scan, etc.). In the implementation, the RLS-controller was made sensitive to the following alerts: attempted Denial of Service, detection of a Denial of Service attack, Denial of Service, and attempted information leak. The latter alert is triggered by TCP `SYN` flooding attacks.

When a DoS attack is identified, the RLS-controller extracts the IP source and destination addresses from the packet provided with the NIDS alert. This IP address pair is then compared to a list of already known pairs that have previously triggered alerts. Each recorded pair identifies a traffic aggregate that is currently rate-limited at the router side. If the new pair is not part of this list, it is added and the RLS-controller sends a new `RateLimit` message to the RLS-agent. It also reports the event to the system administrator via e-mail. If the new pair is already recorded in the list, the attack is already rate-limited and an e-mail has already been sent to the system administrator: any other action is not needed.

6.2.2. The RLS-agent

Like the RLS-controller, the RLS-agent is a user-space program written in C. In the implementation, its function is to translate the messages received from the RLS-controller into TC command lines and to execute them.

Upon initialization, the RLS-agent first replaces the default *First In First Out* (FIFO) queue by a *Class Based Queuing* (CBQ) mechanism. FIFO is the simplest and most common queuing principle: it first sends the packet that first entered the buffer. CBQ “is a hierarchical class-based resource management” used to implement link-sharing and real time services [Flo95]. The Linux implementation of CBQ uses a *Weighted Round Robin* (WRR) scheduler [Rah95] [Kat87] scheduler and a default FIFO queue. WRR is a scheduling mechanism that serves queues in a round after round fashion. Each queue is allotted a weight that represents the amount of service it gets during one turn. Thus, certain connections can be served more than others during the same period of time. The RLS-agent uses two queues: one in which the attack traffic is directed, and another one for the rest of the traffic. Support for only one single attack queue is enough regarding the purposes of the tests. The queue for legitimate traffic is still managed by a default FIFO mechanism while a specific AQM mechanism that randomly drops packets is applied to the attack queue. This AQM mechanism and its properties are described in the next section. At this point, all the components needed to rate-limit traffic are in place, except the filters.

After initialization, the RLS-agent waits for messages from the RLS-controller. Each time a `RateLimit` message is received, the RLS-agent creates a new filtering rule in the TC command line format and executes it. (For example: `tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip dst` *`destination_address`* `match ip src` *`source_address`* `flowid 1:1`). The source and destination IP addresses are filled with the values that the `RateLimit` message provides. After that, each time an incoming packet matches a filtering rule, it is directed into the attack queue.

6.2.3. Communications Between the RLS-controller and the RLS-agent

Communications are a major concern in a real RLS but they have not been really addressed in the implementation. The simplicity and reliability of the environment lead to design a minimal method.

As a distribution protocol, UDP was chosen. It is able to operate over a wide variety of conditions. Although this protocol lacks reliability, neither acknowledgment nor retransmission mechanisms were implemented. The test network offers very good transmission conditions and the probability to lose a message is very low and negligible. UDP was also preferred to TCP because TCP would have slowed down the reaction time because of the initial handshake and the congestion control mechanism. These effects are undesirable in the RLS that should react as fast as possible.

Regarding the content of the messages, solely the `RateLimit` message was implemented with a minimal content: the IP source and destination addresses. Only IP addresses are required to identify the attack traffic aggregate, the other parameters were manually configured in the source code of the RLS-agent. `Cancel` messages were not necessary for the tests and they were not implemented: rate-limiting actions are permanent. As the payload content is very simple, already defined syntax such as the Pushback-messages [Flo01] used in ACC [Mah01] or the messages of the IDIP-protocol [Sch00] were not used: a simple unicast message was enough.

6.2.4. The RLS Specific AQM (RLS-AQM)

To mitigate flooding DoS attacks, it has been proposed in [MDoS, RLS] to constantly drop a certain proportion of the attack traffic. This method allows to preserve the quality of legitimate service whose communications would be misidentified as belonging to an attack. Indeed, this mechanism makes use of the packet loss tolerance of network applications. Thus, the maximum packet loss active legitimate applications can handle is also the maximum proportion of attack packets that can be discarded without disturbing the quality of the services.

To test this rate-limiting method, a new AQM mechanism based on the RED algorithm [Flo93] was designed: the *RLS-AQM*. The only major change is the addition of an initial drop probability *R* that is applied whatever the size of the queue is. The new algorithm keeps the original increasing dropping probability *p* (with a maximum value *P*), but this parameter is nonetheless modified by the additional *R* constant. The shape of the modified dropping probability function is illustrated in the following figure:

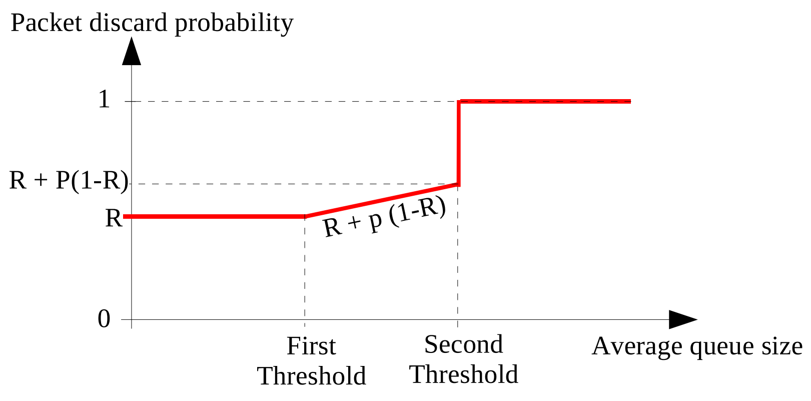

Figure 12: Dropping probability function of the RLS-AQM

The new dropping probability function is then:

R	when average queue size < first threshold
R+ p(1-R)	when first threshold < average queue size < second threshold
1	when second threshold < average queue size

p in the equation *R+p(1-R)* is the increasing drop probability of the normal RED algorithm. Its maximum value *P* is usually small, 0.02, for example. In Linux, implementing the RLS-

AQM requires some simple kernel coding to modify the RED implementation. In the test implementation, the *R* parameter – one subject of the tests described in the next chapter – has to be manually modified in the source file.

As only the initial discard probability (*R*) should be used, the buffer size and the first threshold parameters were configured with high values. In that way, the queue can never grew enough during the tests to exceed the first threshold. Traffic conditions in the network were also planned so that the router (the bottleneck in the system) was never overwhelmed.

6.3. Limitations of the Test Implementation

The implementation does not respect all the requirements to build a real RLS. Only the aspects that could affect the test results, which are mainly focused on the performances of the rate-limiting mechanism, were fully addressed. To build an implementation of the RLS for a real-life environment, several issues have to be taken into account.

As attack detection means are not reliable, some legitimate traffic is expected to suffer rate-limiting. The applications whose communications are rate-limited cannot work at their full potential then. However, a DoS attack can occur without disturbing any service if the target is able to handle it. For example, a weak TCP `SYN` flooding attack may only deny access to half of the port of its target, while only a few of them are normally used. In that case, services are still available and the attack fails. Mitigating the attack does not bring any advantage for users as they are not disturbed by the attack anyway. On the other hand, rate-limiting will damage some legitimate services, would it be very slightly. Then, there is no reason to use rate-limiting. However, this situation is not taken into account by the test RLS. It does not have any knowledge about the effectiveness of detected attacks. Thus, rate-limiting is always applied. This issue was not addressed as it does not affect the effectiveness of rate-limiting, which is the main focus of the tests.

Security is one of the most important concerns in software design. It is even more obvious for the RLS which is meant to be part of a security system. A vulnerability in the program can allow an attacker to shut down the RLS or even to use it as a DoS tool. A particular threat is that an attacker fakes the `RateLimit` message so that it includes the identification pattern of a legitimate traffic aggregate. The authentication of RLS-components and the integrity, confidentiality, and availability of messages are not included in this implementation. Security was not a concern in the tests documented here. The tests were carried out in an isolated, controlled environment therefore there could not be any malicious attempt to disrupt the course of the tests. As for the performances of the system, the reaction delay could be slowed down by security mechanisms (encrypting/decrypting the messages would require some additional time, for example). Then, once rate-limiting takes place, none of the operations changes. Thus, the results of the tests concerning the suitability and performances of rate-limiting would not have been modified.

Knowledge of the rate-limiting status is not implemented neither. the RLS does not have the capabilities on canceling filters. Once a new filter is applied, its action is permanent. Managing up to date information about attacks was out of focus. In a real environment, an attack does not last forever; only the relevant filters have to be maintained. The accuracy of the filtering rules should also be improved and make full use of 5-tuples. It should also be possible to use several queues with different and configurable parameters to adapt the rate-limiting to the type of the traffic. (In the next chapter, tests show the particular importance of this point). Here, the changes require mainly a modification in the communication protocol between the RLS-controller and RLS-agent. No change would be needed for the rate-limiting mechanism, thus, the main results of the tests in this document would not be different.

The test network is quite simple: it includes only one IDS and one RLS-agent. However, a better effectiveness could be achieved by using a diversity of data sources. During the design of the RLS-components, this simplicity also avoids to take into account several issues that a most complex system faces. The scalability of the system, the synchronization and the distribution of information between its components is a critical problem in a real environment. For example, the network to defend may have several routers and/or several interfaces with untrusted networks. Several RLS-agents have to be used in this environment. The point-to-point communication system of the implementation does not suit a highly distributed system. Here again, the changes would not affect the rate-limiting mechanism and the results of the tests in this document would not be modified.

Finally, a RLS user should have the possibility to analyze a record of the actions of the RLS. It should also be possible to correct manually an action which is not appropriate (applying rate-limiting on a well known legitimate service, for example). As no monitoring module has been implemented, the administrator has no more control on the system than starting it and shutting it down. Logs are not recorded neither. This issue is a matter of new functionalities but it does not change the core functionalities that are already integrated in the system. Thus, it does not modify the test results in this study.

7. TESTS, RESULTS, AND ANALYSIS

The main purpose of the tests was to determine the usability and the effectiveness of rate-limiting to mitigate DoS attacks. Therefore, efforts were focused on the effects of RLS on legitimate traffic that is mis-identified as attack traffic, which is the main bound on the utility of rate-limiting.

7.1. Context

To understand the test results, certain important properties of the implementation and the test environment should be stressed [Möl-2].

As the following figure illustrates it, the packet loss in the test network is one-way only. It applies to incoming traffic and messages leaving the protected network do not experience any increased packet loss. When a system suffers a DoS attack, attack traffic is naturally coming towards the target, not leaving from it.

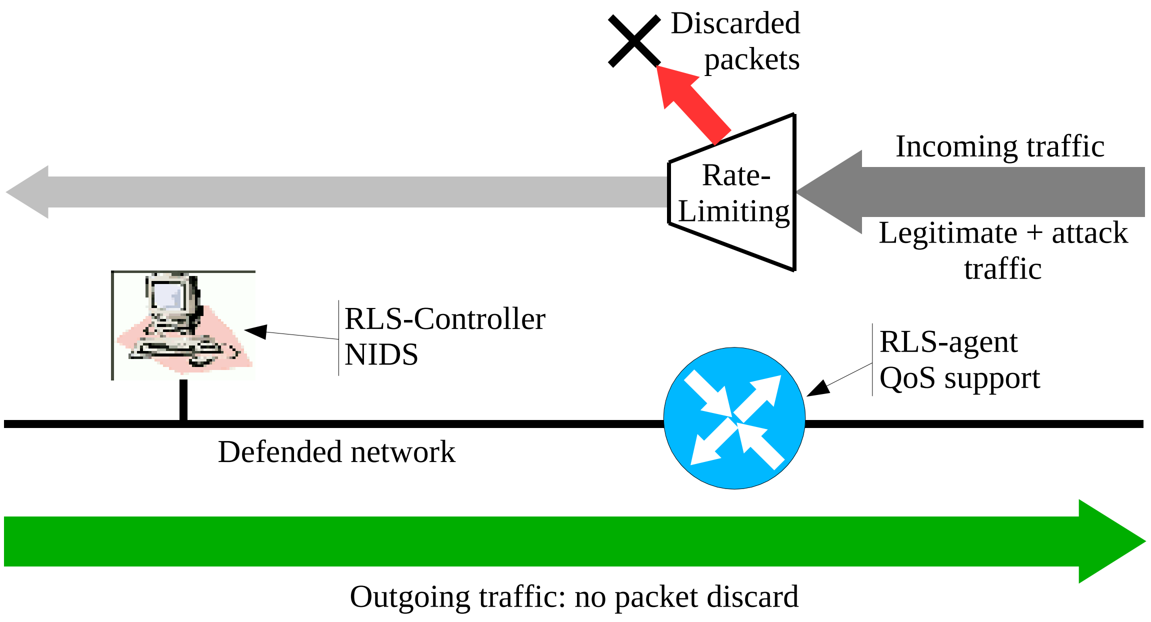

Figure 13: Effects of the RLS on network traffic

Then, the test network represents an almost ideal case. For example, no packet loss was detected while it is inherent to real-life networks. Thus, the tests do not include the effects of random packet losses. The *round-trip time* (RTT) has also an important effect on the way users experience the quality of a network communication. The RTT is the length of time between the sending of a message from a certain host to a second one and the reception of a reply on the first host. It is approximately twice the delay between two hosts. While a delay on the Internet can easily exceed 150 ms, the delay between the most distant hosts in the test network was less than 4 ms. In these conditions, the Linux TCP-implementation uses a small retransmission timeout value: the first retransmission occurs around 200 ms after the first transmission.

7.2. Tests and Results

7.2.1. Accuracy of the Packet Discard Probability

Before assembling the RLS components, the proper functioning of each of them was checked. The RLS-AQM was particularly important as its quality shapes the performance of the system.

To check its behavior, the default FIFO queue on the Network 1 interface of Host 2 (see above: figure 11) was replaced by a queue managed with the RLS-AQM. The queue on the second interface remained the default FIFO that Linux uses. Traffic was then generated between Host 3 and Host 1 using three different applications on Host 3 (HTTP, ping, mstream) to access the services on Host 1. The amount of packets going through the two interfaces of Host 2 in the Host 3 towards Host 1 direction were recorded. Finally, the operation was repeated for different values of the packet discard probability R and the actual packet loss ratios was calculated from the dumped data. The resulting statistics are summarized in the next figure:

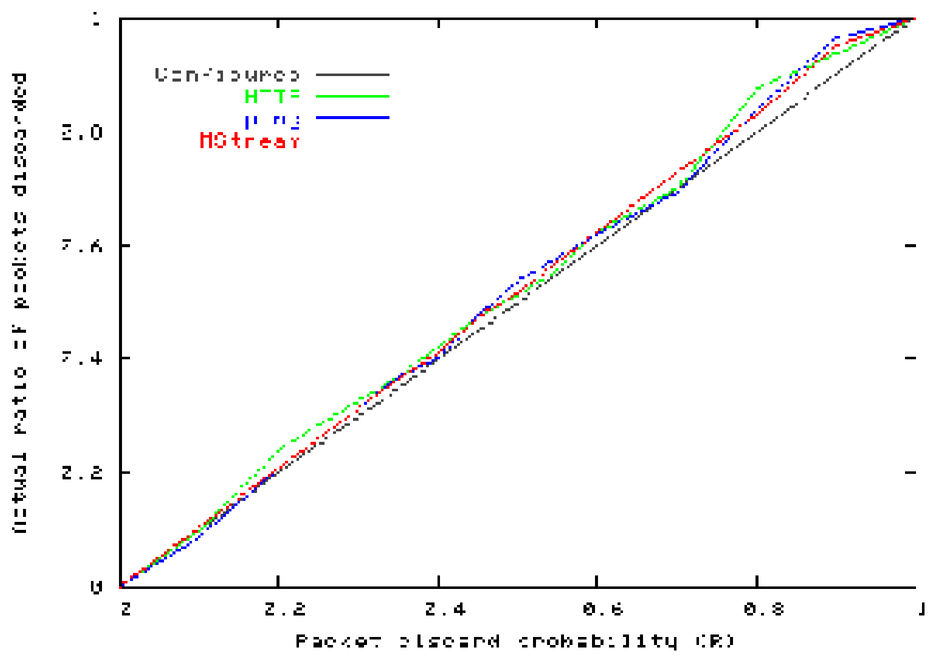

Figure 14: Experienced packet loss ratios compared to configured values

The results show that packet loss ratios are application independent. They also point out that the measured rates are in line with the corresponding configured values, thus validating the proper behavior of the RLS-AQM. However, measured values appeared to be generally slightly above the configured ones, particularly when using high values for discard probability. The difference reaches 7.6% at its maximum. The quality of the random number generator is most probably responsible for these small differences. In the implementation, the Linux kernel function `get_random_bytes()` was used to provide a one-byte random unsigned integer (256 possible values).

7.2.2. Global Performances of the System

Naturally, the proper functioning of the whole RLS had to be validated before proceeding to any further test. This step also provided some rough data about the performances of the system.

7.2.2.1. Testing the proper behavior of the RLS

Like the RLS-AQM, the RLS-controller and RLS-agent were also separately tested. Once the final versions were validated, the correct operation of the whole system was checked. Having started the three components of the RLS (IDS, RLS-controller and RLS-agent), traffic was generated between the two end hosts with mstream and two instances of the Mozilla web-browser on Host 3 (figure 11). The mstream agent and one instance of Mozilla were using the Network 1b address of Host 1 while the second Mozilla instance was accessing the web-server on the Network 1a address of Host 1. The RLS-AQM packet discard ratio was set to 0.5. Packets going through the two interfaces of Host 2 in the Host 3 to Host 1 direction were recorded.

Not using mstream, no packet loss was observed. When mstream is in action, traffic from mstream and from the web-browser using the Network 1b address experience a close to 50% packet loss. One record shows 52.6% packet discard for mstream traffic and 55.6% for web-browsing (with only 180 packets sent). The packet losses cannot be due to the DDoS tool as its capabilities are reduced in the source code. It cannot cause any overload in the network nor can it consume all of its target port: services are still available. In addition, a warning e-mail message is delivered to the system administrator.

These results show that the system behaves as it should. Upon attack, the IDS recognizes the mstream packets and it alerts the RLS-controller. The latter classifies packets coming from Host 3 and directed to the Network 1b address of Host 1 as belonging to an attack. Therefore, the legitimate traffic generated by the web-browser using the Network 1b address of Host 1 is amalgamated with the attack traffic aggregate. When the RLS-agent receives the `RateLimit` message, it creates the relevant filter. After that, both attack and mis-identified legitimate traffic are directed in the attack queue and suffer packet losses. The legitimate traffic using the Network 1a address of Host 1 is not selected by the filter and passes through Host 2 without being rate-limited.

7.2.2.2. Attack detection and reaction delay

A second test, focusing on the reaction delay, also demonstrates the correct behavior of the RLS.

The system was ran with an initial drop probability of the RLS-AQM set at 100%. In that way, as soon as the rate-limiting occurs, the detected attack traffic is blocked. The traffic was captured with tcpdump on the Network 1 physical interface of Host 2 (figure 11). The RLS was first started. Then, attack traffic was generated with the mstream agent on Host 3. The length of time between the first and the last attack packets recorded provided the reaction delay of the RLS. Attack packets are easily identifiable as all of them present this pattern: `ack 0 win 16384 [tos 0x8]`. This test was ran several times to ensure the validity of the measurements.

The results show a delay around 75ms. Notice that the network delay between Host 1 and Host 2 is typically on the order of 0.25 ms only (3,9 ms when using 1450 byte long packets) .

Except rate-limiting that is performed by the Linux kernel, most of the RLS components of the implementation we described are programs running in the user space (TC, IDS, RLS-controller, RLS-agent) so the reaction delay value is not surprising. This delay could be improved using programs running in the kernel space but the present reaction delay of the system is already satisfactory. However, several tens of attack packets go through Host 2 before the RLS reacts (not forgetting that mstream has been rate-limited by modifying the source code). This proves the weakness of the system facing a logic DoS attack (see section 2.1.2. above, [Moo01]), even using blocking instead of rate-limiting: it cannot be stopped on time.

7.2.3. Effects of the RLS on the Usability of TCP Applications.

This is the main test that was carried out. It roughly determined the usability bounds of the RLS, as specified in [Möl-2].

7.2.3.1. Measurements in a one-way rate-limited network

To be efficient, the RLS should maximize the quantity of attack traffic discarded while minimizing harm on legitimate traffic [MDoS, RLS]. The upper bound on the value of the packet discard ratio R is reached when it starts to damage the usability of a service. Therefore, an exact upper value for R cannot be determined objectively; it depends on how a user perceives the quality of the service. Anyway, the tests that were performed provided some good indications about the usability of rate-limiting.

The effects of rate-limiting were tested on three common TCP-based services: HTTP, FTP-downloading, and FTP-uploading. All these tests were carried out by configuring the test network so that the RLS-AQM is active on the Network 1 interface of Host 2 (figure 11). This triggers the same one-way packet discard as in any attack queue when the RLS is on (see figure 13). The measurements were carried out with several different values of packet discard probability. No background traffic was generated.

The HTTP test was carried out in a simple way. A tester used the web-browser on Host 3 to access a website located in the web-server on Host 1. For different values of packet discard probability, the tester browsed the website and recorded its impressions. Naturally, this test did not provide any objective measurements: results only rely on the appreciations of the tester. The usability of web-browsing is highly subjective so the reliability of the measurements cannot be guaranteed. From the results, it appeared that the first negative effects can be perceived around R = 0.3 but the quality of web-browsing remains acceptable until R = 0.55.

In the FTP case, the client was ran on Host 3 to access the server on Host 1. (Later tests in the reverse configuration showed similar results.) The FTP-client provided the transfer rate when downloading or uploading a 450 kbyte-file. The results are shown in the figure 15 and 16 hereafter, in which the theoretical throughput for a TCP application defined by the equation [Mat97]

$$BW = \frac{MSS}{RTT}\frac{C}{\sqrt{p}} \qquad (1)$$

is also represented. This simple equation is only valid for small values of the packet discard ratio, which is enough here. One can refer to [Pad98] for a more accurate theoretical model. The curve helps to understand the particularities of the following measurements. The parameters of the equation were set as follow: MSS = 1448 bytes, RTT = 3,9 ms and C = 0.00022. MSS designates the maximum segment size that the network can handle without fragmenting packets. The RTT has been measured with the ping application using 1450 byte long ICMP messages. Finally, C represents a constant of proportionality whose value was determined by testing. In the figures 15 and 16, the blue line corresponds to the theoretical model. The red line represents the measured average throughput and the gray vertical bars are error bars.

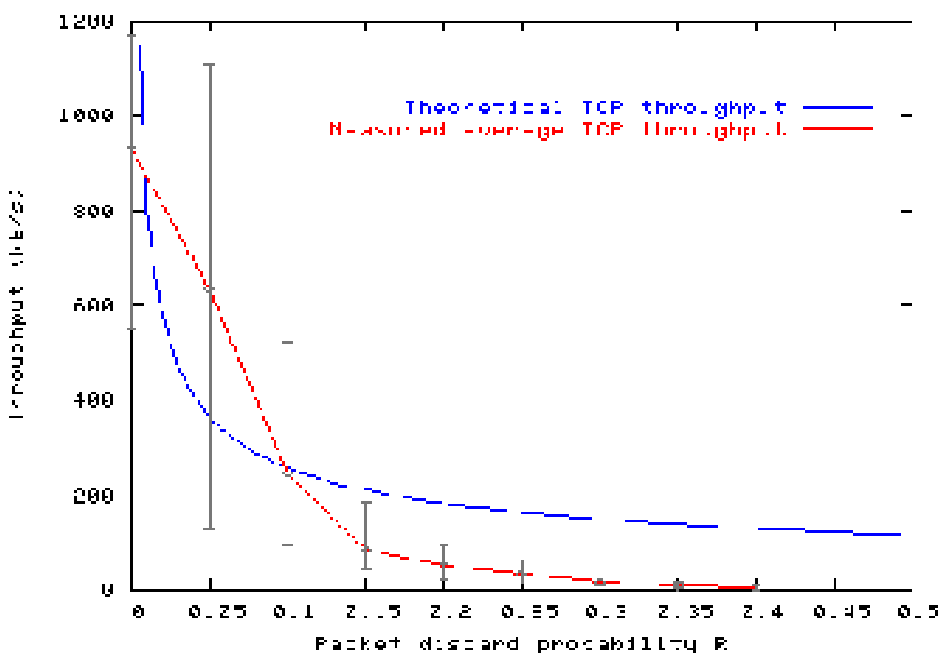

Figure 15: FTP-uploading rates for different packet discard probability values in a one-way rate-limited network.

Looking at the FTP-uploading graph, the random aspect of the packet discard does not seem to affect very much the overall behavior of TCP: if values are quite scattered at the beginning they quickly narrow. The shape of the throughput itself follows quite well the theory. Effect of rate-limiting are very destructive: the transfer rate decreases exponentially when *R* increases. *R* = 0.05 already halves the average throughput. A value larger than 0.1 does not seem to be useful in practice. This is largely in line with the theoretical evaluation of TCP throughput according to the equation (1) [Mat97].

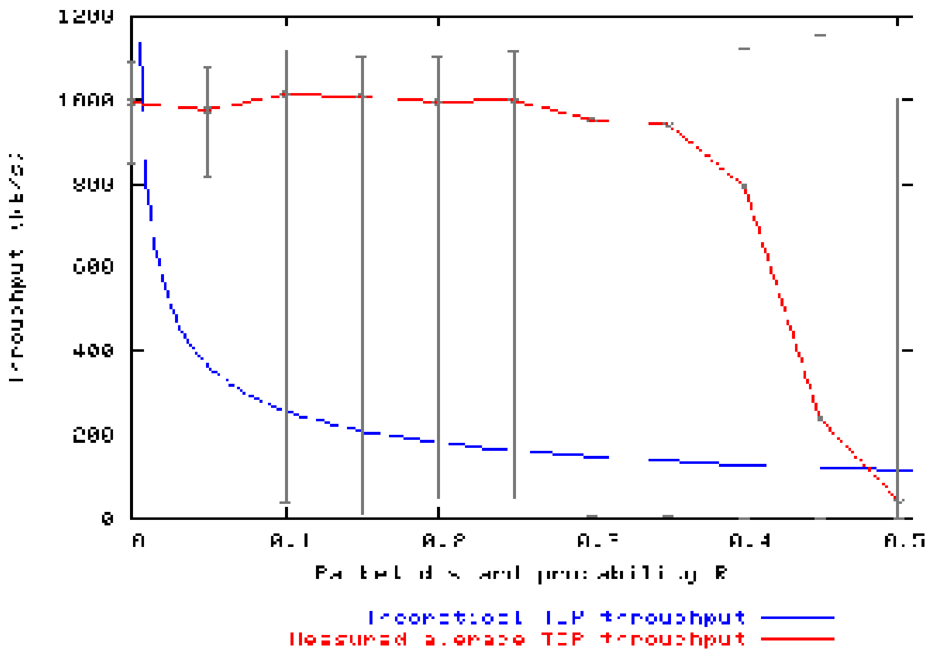

Figure 16: FTP-downloading rates for different packet discard probability values in a one-way rate-limited network.

Concerning the FTP-downloading traffic, the shape of the curve does not follow at all the theory. Measurements are much better and throughput really starts to decrease after *R* = 0.35. However, the error bars show that some very small throughput values occur sometimes, even with a quite high average. The next figure gives a better idea of this phenomenon. The measured average throughput is still represented in red and each green diamond corresponds to one measurement.

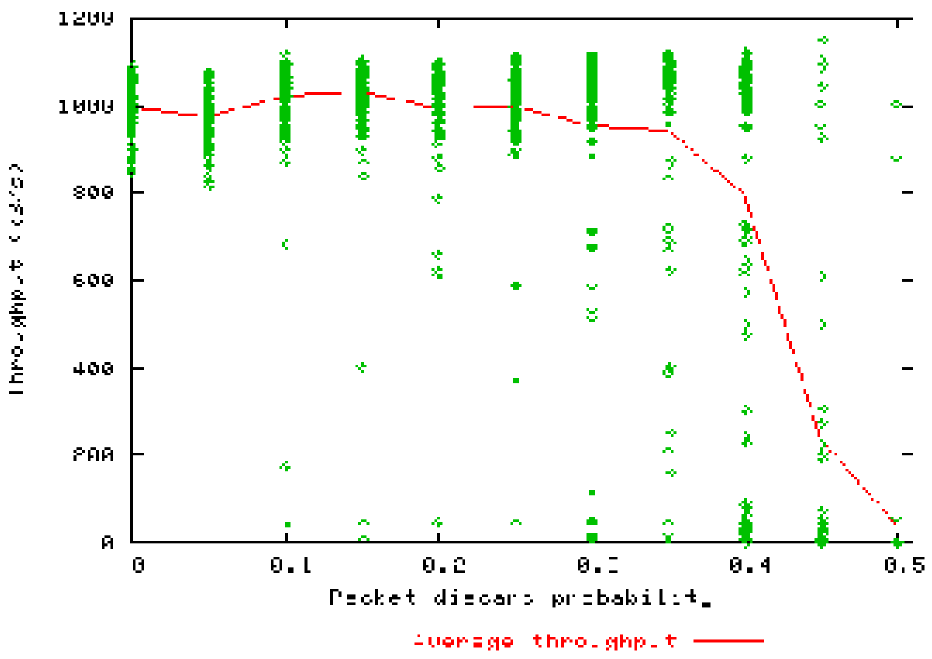

Figure 17: Dispersion of measurements for FTP-downloading with a constant file size (450 kB)

The figure 17 is the same than the figure 16 but all the samples are displayed. This gives a better idea of the distribution of the values. One can see that measurements are pretty concentrated around the average value with low values of *R*. Then, little by little, samples start to spread while *R* increases. Finally, the concentration of measurements shifts steeply from high values to low ones from *R* = 0.40, triggering the steer decreasing shape of the curve.

The figure 18 shows all the measurements that were done in the FTP-downloading case with *R* = 0.25 while using different file sizes. These measurements were carried out to check if the results were affected by the file size. The average throughput is represented by the red line and the measurements are displayed as green diamonds.

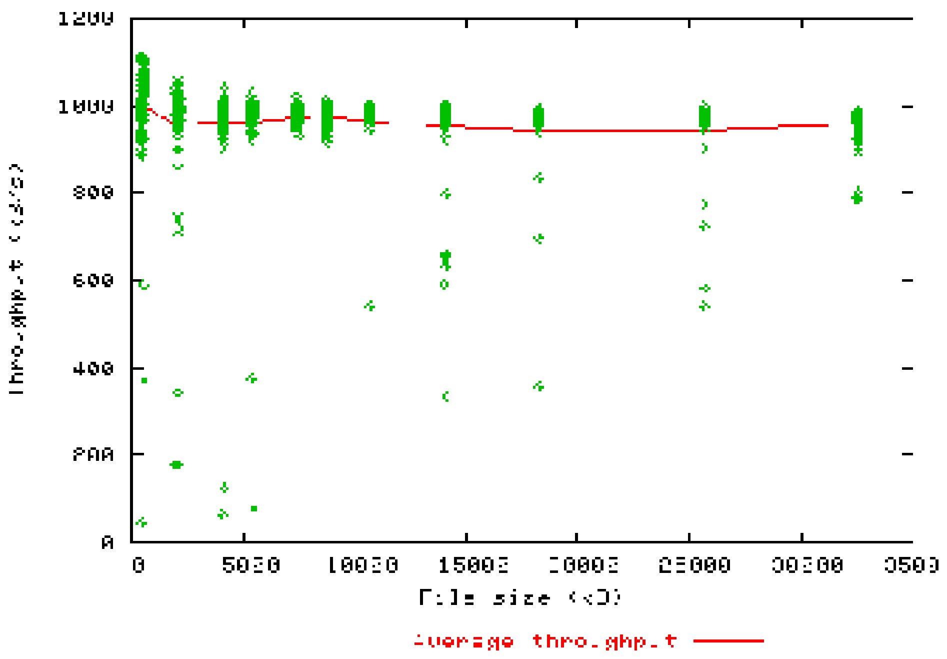

Figure 18: Dispersion of measurements for FTP-downloading with a constant R = 0.25

For each file size value, almost all the measurements are grouped together in a tight zone around the actual average. This ensures the validity of the previous results when considering them in a general case where file size varies. Only a few measurements have very different values than the average. In theory, one can state that the throughput can even reach 0 kB/s from the moment when $R > 0$. In practice, one can notice that the larger the file size is, the less frequent very small throughputs are.

7.2.3.2. Measurements using the whole RLS

The previous results were confirmed when using the whole RLS system as described in [Möl-2]. In that test, the RLS was first initiated: the NIDS, the RLS-controller, and the RLS-agent were started. A DoS attack targeting Host 1 was then launched from Host 3 using the Network 1b destination address (see figure 11). This action was triggering off permanent rate-limiting of traffic coming from Host 3 and going to the Network 1b address of Host 1. The attack was stopped after a short while and the FTP transfer tests were carried out using the Network 1b address of Host 1 too.

These conditions are equivalent to the test conditions used in the previous section and the same results were indeed obtained. The next figure shows the measurements. As in figure 15 and 16, the blue curve represents the theoretical model defined by the equation (1)

[Mat97]. The red line represents the measured average throughput and the gray vertical bars are error bars.

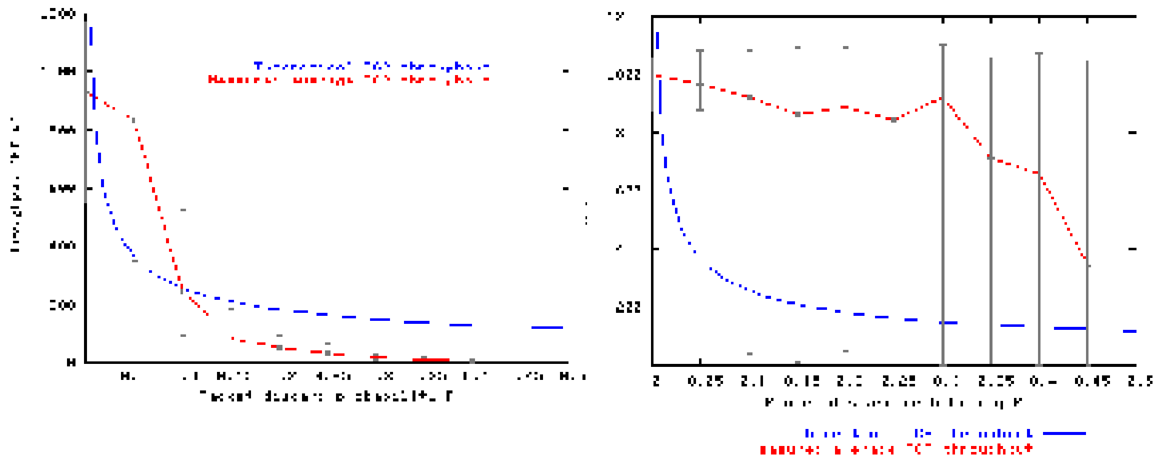

***Figure 19: FTP-uploading and downloading rates for different packet discard probability values from* [Möl-2]**

The difference between the look of these curves compared to the previous measurements is due to several minor factors such as the amount of samples that were obtained from the tests (higher number in the figures 15 and 16 above), the randomness of the results, and the performance of the traffic control implementation in the router (using the RLS adds traffic classification and a scheduler).

7.2.3.3. Analysis

The measurements clearly show that the effects of the RLS are highly application dependent. This is not surprising as applications do not have the same properties in term of packet-loss tolerance. Packet loss is less critical in an application like HTTP than in FTP.

The FTP-uploading/-downloading case is more relevant concerning the most interesting property of the RLS. The source of the difference between the two curves is the kind of FTP messages that are discarded. As rate-limiting is only applied in one direction, only data packets or acknowledgments are discarded. When data packets are dropped, the TCP protocol requires to retransmit them. Otherwise data would be lost. It results in a weak tolerance to packet loss as the FTP-uploading case illustrates. Indeed, when uploading a file from Host 3 to Host 1 (see figure 11), data packets are going through the rate-limiting system and acknowledgments are leaving the network without suffering any packet discard. On the contrary, in the FTP-downloading case, only acknowledgments are discarded. Unlike the data packets, TCP acknowledgments do not necessary need to be retransmitted; the lost information can be recovered by following acknowledgments. Thus, FTP-downloading requires far less retransmissions than FTP-uploading and the transfer conditions remain good with packet discard ratios much higher than what FTP-uploading can withstand. This also explains the difference with the theoretical curve from [Mat97] that represents the behavior of TCP when data packet are lost; the model does not take into account acknowledgment retransmissions. While there is a high amount of literature related to the behavior of TCP when data packets are lost, there are a few studies about the behavior of TCP when acknowledgment are lost. Concerning the RLS, one can only regret it because such studies would be very relevant.

7.3. Conclusions on the Usability of the RLS

When using a TCP application, the RLS preserves traffic better when acknowledgments are rate-limited and data packets are transmitted in the downward direction, away from the RLS-protected site. Therefore, the relevance of applying RLS while using a certain application is dependent on the characteristics of the traffic that the application generates.

From the tests, it appears that RLS can discard up to 40% of DoS attack packets while not damaging the average quality of a HTTP or FTP-downloading service. The higher the discard probability is, the higher the probability that some users occasionally experience very low service quality is. These two services are the most common services offered by websites on the Internet – including e-commerce sites. A typical Internet user usually downloads large figures, files, web-pages, streaming video, etc. from servers and seldom uses the opposite operation. Moreover, well-known websites have been a target for many published flooding DoS attacks [Gar00]. Thus, RLS is a suitable automatic mechanism to defend websites against low bandwidth flooding attacks. Such attacks are common attacks and are often performed using methods like TCP `SYN` or ICMP `Echo Request` flooding. A study showed that 12000 DoS attack were carried out in the Internet against more than 5000 different targets during a three weeks period in the beginning of year 2001. More than 90% of these attacks were TCP-based and TCP `SYN` flooding was thought to be the most likely method attackers were using [Moo01].

In a real environment, legitimate flows are probably less tolerant than the above measurements. The packet loss inherent to real-life environment and the high delay values were not taken into account in the small test network. Therefore, one can expect smaller upper *R* values to be more realistic. Anyway, the test results provided some relevant indications on the usability of the RLS. Further studies are needed to get more precise and exhaustive results.

7.4. Further Issues Regarding the RLS

This document presented a study focused on implementing and testing rate-limiting as an automatic defense mechanism against DoS attacks. The objectives were to validate the idea, not to produce an optimal system. To reach this other goal, several topics have to be deepened.

7.4.1. A Complete System

As stated in the section 6.3., the test implementation was not designed for a real-life use. It was designed as a minimum implementation for the purpose of a few tests. This implementation was enough to validate the suitability of rate-limiting. The following step would be to realize a complete implementation addressing all the issues to which the RLS can be confronted: distribution of information, security mechanisms, reliable recording and follow-up of actions, and so on. Once all the issues are solved, one could also get a better idea of the costs of an implementation in a real-life environment. This is also an important element when deciding if the RLS is worth to be used to defend a real system.

7.4.2. Measurements

The results used in this thesis can be improved regarding several aspects. First, the test network was not reflecting the complexity and the conditions of real-life environments. Only three stations were used in a small local area network. A more realistic environment including remote hosts using the Internet would be particularly relevant to confirm the results. In a final step, a real-life test should be performed using the RLS to defend a real system, such as a well-known website. In each case, all meaningful data should be logged to validate the behavior of the system afterwards.

Moreover, the tests did not allow to find the accurate limits of the system. It would be particularly interesting to determine the best packet discard probability values that can be used. This requires more exhaustive testing, particularly regarding the amount of measurements. The methods used during the tests can also be improved. Some objective indications could be obtained from the HTTP test by measuring the length of time between the page requests and the display of the page itself. Still on the same test, a good evaluation of the performances would require the participation of a large pool of users to get some precise statistics.

In this thesis, we only studied the effect of rate-limiting on three applications (HTTP, FTP-uploading, and FTP-downloading) that were all using the same transport protocol: TCP. This was enough to find one area of application for the RLS; it can be used to defend web-servers. Studying the effects of one-way packet loss on other applications and on different transport protocols would allow to find other areas of application and to which extent it can be used in the context of the current area of application.

7.4.3. Queue management

The test RLS described above does only include one attack queue and one legitimate queue. According to the specifications in [Möl-1], the RLS should be able to make use of several queues using parameters adapted to the traffic they receive.

The test RLS does only select traffic based on one criteria: whether the traffic is considered legitimate or belonging to an attack. Then, the source and destination IP addresses are used to direct the traffic flows into the legitimate or attack queue. Thus, only two different queues are used while in the theoretical requirements the possibility to create several attack queues was mentioned. These queues would experience different traffic condition, according to a more precise characterization of the kind of traffic they receive. For example, the traffic could be split according to the transport protocol in use, or the application that uses it, and so on. The traffic could also be identified more precisely; in [Möl-2] the five-tuple is indicated to be a good trade-off between the precision of the identification and the time needed to process the traffic flows.

Using the current RLS-AQM, it can also be possible to automatically change the value of its initial drop probability according to the actively used services. For example, if the users of a web-server do only use the HTTP service, the RLS-AQM could be configured to discard 45% of the traffic it receives. Then, learning that the FTP service is used, the discard ratio could be decreased to 35%. In another case, learning that the attack pattern is changing for a high bandwidth attack, the RLS-AQM could discard every packet it receives. These changes require the RLS to work in a completely automatic and fast-reacting way. Some particular difficulties could be expected when bringing the RLS to operate closer to real-time and when improving the knowledge of the RLA about the defended system and the effects of attacks and rate-limiting on it.

The generic idea implies gathering information from the defended system and having different AQM mechanisms to manage the queues or at least different AQM parameters for each queue. But how to configure these parameters? Each attack queue receives two kind of traffic: attack traffic and legitimate traffic. If the properties of the attack can be known (thanks to the detection system), the kind of mis-identified legitimate traffic cannot be reliably identified. Unfortunately, harm on legitimate traffic determines the upper rate-limiting bounds of the system. Optimization of the queuing policy according to the selected attack/legitimate traffic is a hard problem.

Moreover, what about implementing several legitimate queues? If the network supports QoS, there is a need for a normal use of traffic control. Currently, the solution would be to operate QoS operation on traffic going out from the legitimate queue. But the RLS-agent would be more efficient if it was QoS-capable since it would avoid additional delay due to the buffer of the RLS.

7.4.4. Communication Protocols

The implementation uses a unicast message sent over UDP but this is not be the most relevant proposal in the general case (see sections 5.5.2. and 6.2.3. above). What is the best transport protocol/payload content couple? This remains an open question and it has direct effects on the performance of the system. Therefore, in order to improve the RLS, this is one important issue that should be studied.

As specified in the chapter 5. about the design requirements, the best possible communication protocol should be reliable, real-time, unicast and multicast capable, scalable, secure, and the most efficient possible. Some functions can be operated by the transport protocol, the rest should be taken into account in the payload. For example, if UDP is used as a transport protocol, then the messages should include an acknowledgment mechanism to make up for the lack of reliability. If using already existent proposals would avoid reinventing the wheel, designing a brand new protocol is still a valid option while it cannot be ensured that an existing protocol fits the requirements.

8. CONCLUSION

As the survey showed, DoS attacks are currently one of the most important problems in computer network security. Even if some tools like IDSs can help to defend systems against DoS attacks, there is no comprehensive solution yet. This thesis documented the RLS, described in [Möl-1] and [Möl-2], which is a system that can improve the existing defense mechanisms. Its principle is to mitigate DoS attacks using a rate-limiting mechanism in order to avoid damaging legitimate services. The goals of two existing proposals, ACC [Mah01] and CITRA [Sch01], were close to the RLS principle but these proposals did not study in depth the side effects of the defense mechanisms on legitimate traffic. This document aimed to address that issue by providing the description and the test results of a simple implementation of the RLS.

The implementation of the RLS was carried out in a Linux environment and tested in a small network. The design and the tests focused on the important principles in order to validate the rate-limiting concept and its suitability to mitigate DoS attacks. In the RLS, rate-limiting is only applied on traffic entering the defended network. The rate-limiter of the system is the RLS-AQM, which is an AQM mechanism whose function is to randomly drop a certain proportion of packets.

In a one-way packet loss network using the RLS-AQM, the tests showed that the HTTP service was able to withstand a 55% packet drop rate and the FTP service less than 10% when uploading, and up to 40% when downloading. The same results were also discussed in [Möl-2]. The suitability of rate-limiting is, thus, application dependent and important usability differences can occur according to the kind of packets that suffer rate-limiting. In the FTP case, when uploading, data packets are discarded and every discarded packet has to be retransmitted. On the contrary, when downloading, only TCP acknowledgments are discarded and they do not necessarily need to be retransmitted, as the following acknowledgments can make up for the lost information.

The tests showed the suitability of rate-limiting to defend a system in a precise context, mainly against low bandwidth flooding attacks such as a typical TCP `SYN` or ICMP `Echo Request` flooding attack. Rate-limiting was shown to be an effective mechanism to defend a system that offers HTTP and FTP-downloading services, for example. This is a particularly interesting area of application because websites, which are common victims of flooding DoS attacks, are mainly providing these services.

Nonetheless, rate-limiting is only one particular defense mechanism and it should be used as part of a comprehensive security architecture including IDSs, firewalls, antivirus software, etc. The RLS is only useful in a precise context that can even seem quite restricted. Moreover, further research is needed to improve the RLS and to validate the rate-limiting principle in more realistic environments. The values presented here can be considered as best case values because of the characteristics of the test environment.

REFERENCES

[All00] J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, E. Stoner. (January 2000). “State of the Practice of Intrusion Detection Technologies”. Software Engineering Institute. Pittsburgh, Pennsylvania, USA. [Online]. Available: http://www.cert.org/archive/pdf/99tr028.pdf

[Bra94] R. Braden, D. Clark, S. Shenker. (1994, June). “Integrated Services in the Internet Architecture: an Overview”. RFC 1633. IETF Network Working Group. [Online]. Available: http://www.ietf.org/rfc/rfc1633.txt

[Bra97] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin. (1997, September). “Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification”. RFC 2205. IETF Network Working Group. [Online]. Available: http://www.ietf.org/rfc/rfc2205.txt

[Bel03] S. Bellovin. (2003, April 1). “The Security Flag in the IPv4 Header”. RFC 3514. IETF Network Working Group. [Online]. Available: http://www.ietf.org/rfc/rfc3514.txt

[CER97] Computer Emergency Response Team. (1997, September 24). “UDP Port Denial-of-Service Attack”. CERT Coordination Center Advisory CA-1996-01. [Online]. Available: http://www.cert.org/advisories/CA-1996-01.html

[CER00] Computer Emergency Response Team. (2000, March 3). “Denial of Service Tools”. CERT Coordination Center Advisory CA-1999-17. [Online]. Available: http://www.cert.org/advisories/CA-1999-17.html

[CER00-1] Computer Emergency Response Team. (2000, March 13). “Smurf IP Denial of Service Attacks”. CERT Coordination Center Advisory CA-1998-01. [Online]. Available: http://www.cert.org/advisories/CA-1998-01.html

[CER00-2] Computer Emergency Response Team. (2000, May). “mstream Distributed Denial of Service Tool”. CERT Coordination Center Incident Note IN-2000-05. [Online]. Available: http://www.cert.org/incident_notes/IN-2000-05.html

[CER00-3] Computer Emergency Response Team. (2000, September 15). “Widespread Exploitation of rpc.statd and wu-ftpd Vulnerabilities”. CERT Coordination Center Incident Note IN-2000-10. [Online]. Available: http://www.cert.org/incident_notes/IN-2000-10.html

[CER00-4] Computer Emergency Response Team. (2000, November 29). “TCP SYN Flooding and IP Spoofing Attacks”. CERT Coordination Center Advisory CA-1996-21. [Online]. Available: http://www.cert.org/advisories/CA-1996-21.html

[CER01] Computer Emergency Response Team. (2001, January 15). "Distributed Denial of Service Tools". CERT Coordination Center Incident Note IN-99-07. [Online]. Available: http://www.cert.org/incident_notes/IN-99-07.html#tfn

[CER01-1] Computer Emergency Response Team. (2001, January 15). "Distributed Denial of Service Tools". CERT Coordination Center Incident Note IN-99-07. [Online]. Available: http://www.cert.org/incident_notes/IN-99-07.html#trinoo

[CER01-2] Computer Emergency Response Team. (2001, August 6). "Code Red II:" Another Worm Exploiting Buffer Overflow In IIS Indexing Service DLL". CERT Coordination Center Incident Note IN-2001-09. [Online]. Available: http://www.cert.org/incident_notes/IN-2001-09.html

[CER02] Computer Emergency Response Team. (2002, January 17). " "Code Red" Worm Exploiting Buffer Overflow in IIS Indexing Service DLL". CERT Coordination Center Advisory CA-2001-19. [Online]. Available: http://www.cert.org/advisories/CA-2001-19.html

[Chu02] T-A. Chut. (2002, March 1). "Traffic Management Part 2: Pipelining Specifics" in *Communication Systems Design*. [Online]. Available: http://www.commsdesign.com/design_corner/OEG20020301S0022

[Cop99] J. A. Copeland. (1999, December 22). "The "Mac DoS Attack," a Scheme for Blocking Internet Connections". Georgia Tech ECE. Atlanta, Georgia, USA. [Online]. Available: http://www.csc.gatech.edu/macattack/macattack.html, http://people.atl.mediaone.net/jacopeland

[Cow01] J. Cowie, A. T. Ogielski, B. J. Premore, and Y. Yuan. (2001, December). "Global Routing Instabilities Triggered by Code Red II and Nimda Worm Attacks". Renesys Corporation. Hanover, New Hampshire, USA. [Online]. Available: http://www.renesys.com/projects/papers/renesys_bgp_instabilities2001.pdf

[Dar00] T. Darmohray, and R. Oliver. (2000, July). " "Hot Spares" for DoS Attacks" in *;login: The Magazine of USENIX and SAGE*, vol. 25, no. 4. [Online]. Available: http://www.usenix.org/publications/login/2000-7/apropos.html

[Dit99] D. Dittrich. (1999, October 21). "The DoS Project's "trinoo" Distributed Denial of Service Attack Tool". University of Washington. Seattle, Washington, USA. [Online]. Available: http://staff.washington.edu/dittrich/misc/trinoo.analysis

[Dit99-1] D. Dittrich. (1999, October 21). "The "Tribe Flood Network" Distributed Denial of Service Attack Tool". University of Washington. Seattle, Washington, USA. [Online]. Available: http://staff.washington.edu/dittrich/misc/tfn.analysis.txt

[Dit99-2] D. Dittrich. (1999, December 31). “The “Stacheldraht” Distributed Denial of Service Attack Tool”. University of Washington. Seattle, Washington, USA. [Online]. Available: http://staff.washington.edu/dittrich/misc/stacheldraht.analysis

[Dow03] M. Dowd and N. Mehta. (2003, March 3). “Snort RPC Preprocessing Vulnerability”. Security advisory. Internet Security Systems X-Force. Atlanta, Georgia, USA. [Online]. Available: http://www.iss.net/issEn/delivery/xforce/alertdetail.jsp?oid=21951

[Fer00] P. Ferguson, and D. Senie. (2000, May). “Network Ingress Filtering: Defeating Denial of Service Attacks which Employ IP Source Address Spoofing”. IETF Network Working Group. [Online]. Available: http://www.ietf.org/rfc/rfc2827.txt

[Flo93] S. Floyd and V. Jacobson. (1993, August). “Random Early Detection Gateways for Congestion Avoidance” in *IEEE/ACM Transactions on Networking*, vol. 1, issue 4, pp. 397-413. Lawrence Berkeley Laboratory, University of California. Berkeley, California, USA. [Online]. Available: http://www.icir.org/floyd/red.html

[Flo95] S. Floyd, and V. Jacobson. (August 1995). “Link-Sharing and Resource Management Models for Packet Networks”, in *IEEE/ACM Transactions on Networking (TON)*, vol. 3, issue 4.

[Flo01] S. Floyd, S. Bellovin, J. Ioannidis, K. Kompella, R. Mahajan, and V. Paxson. (2001, July). “Pushback Messages for Controlling Aggregates in the Network”. Internet-Draft draft-floyd-pushback-messages-00.txt, work in progress. [Online]. Available: http://www.icir.org/pushback, http://www.icir.org/floyd/papers.html

[Gar00] L. Garber. (2000, April). “Denial-of-Service Attacks Rip the Internet” in *IEEE Computer*, vol. 33, no. 4, pp.12-17.

[Hai03] J, Haines, D. K. Ryder, L. Tinnel, and S. Taylor. (2003, January/February). “Validation of Sensor Alert Correlators”, in *IEEE Security & Privacy*, vol. 1, no. 1, pp. 46-56.

[Hei99] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. (1999, June). “Assured Forwarding PHB Group”. RFC 2597. IETF Network Working Group. [Online]. Available: http://www.ietf.org/rfc/rfc2597.txt

[Hei99-1] J. Heinanen, and R. Guerin. (1999, September). “A Single Rate Three Color Marker”. RFC 2697. IETF Network Working Group. [Online]. Available: http://www.ietf.org/rfc/rfc2697.txt

[Hei99-2] J. Heinanen, and R. Guerin. (1999, September). “A Two Rate Three Color Marker”. RFC 2698. IETF Network Working Group. [Online]. Available: http://www.ietf.org/rfc/rfc2698.txt

[Hou01] K. J. Houle, G. M. Weaver, N. Long, and R. Thomas. (2001, October). "Trends in Denial of Service Attack Technology". CERT Coordination Center. [Online]. Available: http://www.cert.org/archive/pdf/DoS_trends.pdf

[Hub03] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. B. Schroeder, J. Spaans, and P. Larroy. (2003, July 26). "Linux Advanced Routing & Traffing Control HOWTO", work in progress. [Online]. Available: http://lartc.org/#download

[Jac99] V. Jacobson, X. Nichols, and K. Poduri. (1999, June). "An Expedited Forwarding PHB". RFC 2598. IETF Network Working Group. [Online]. Available: http://www.ietf.org/rfc/rfc2598.txt

[Kat87] M. G. H. Katevenis. (1987, October). "Fast Switching and Fair Control of Congested Flow in Broadband Networks" in *IEEE Journal on selected Areas in Communications*, vol. SAC-5, no. 8, pp 1315-1326.

[Lai00] B. Laing. (2000). "Implementing a Network Based Intrusion Detection System". Internet Security Systems. [Online]. Available: http://www.snort.org/docs/iss-placement.pdf

[Lip00] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R.K. Cunningham, and M. A. Zissman. (2000, January 25-27). "Evaluating Intrusion Detection Systems: the 1998 DARPA Off-line Intrusion Detection Evaluation", in *Proceedings of the DARPA Information Survivability Conference and Exposition, 2000 (DISCEX '00)*, vol. 2, pp. 12-26.

[Lun03] J. van Lunteren and T. Engbersen. (May 2003). "Fast and scalable packet classification" in *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, pp 560-571.

[Mah01] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. (2001, July 13). "Controlling High Bandwidth Aggregates in the Network (Extended Version)". Draft paper pushback- Jul01.ps, work in progress. AT&T Center for Internet Research at ICSI (ACIRI) and AT&T Labs Research. [Online]. Available: http://www.icir.org/pushback

[Man02] C. Manikopoulos, and S. Papavassiliou. (October 2002). "Network Intrusion and Fault Detection: a Statistical Anomaly Approach" in *IEEE Communications Magazine*, vol. 40, issue 10, pp. 76-82.

[Mat97] M. Mathis, J. Semke, and J. Mahdavi. (1997). "The Macroscopic Behavior of the TCP congestion avoidance algorithm", in *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 3.

[McC03] B. McCarty. (2003, July-August). "Botnets: Big and Bigger" in *IEEE Security & Privacy Magazine*, vol. 1, issue 4, pp 87-90.

[Moo01] D. Moore, G. M. Voelker, and S. Savage. (2001, August). “Inferring Internet Denial of Service Activity”, in *Proceedings of the 10th USENIX Security Symposium*. Washington D.C., USA.

[Möl-1] J. Mölsä, “Mitigation of Denial of Service Attacks”, submitted.

[Möl-2] J. Mölsä, E. Guiton. “Rate-limiting as an Automatic Reaction Mechanism against Flooding DoS Attacks”, submitted.

[Nic98] K. Nichols, S. Blake, F. Baker, and D. Black. (1998, December). “Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers”. RFC 2474. IETF Network Working Group. [Online]. Available: http://www.ietf.org/rfc/rfc2474.txt

[Pad98] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. (1998, September). “Modeling TCP Throughput: A Simple model and its Empirical Validation”, in *Proceedings of the ACM SIGCOMM conference*. Vancouver, Canada.

[Pos81] J. Postel. (1981. September). "Internet Control Message Protocol”, *DARPA Internet Program Protocol Specification*. RFC 792. USC/Information Sciences Institute. [Online]. Available: http://www.ietf.org/rfc/rfc0792.txt

[Pta98] T. H. Ptacek and T. N. Newsham. (1998, January). “Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection”. Technical Report, Secure Networks, Inc. [Online]. Available: http://www.securityfocus.com/data/library/ids.ps

[Rah95] A. Raha, N. Malcolm, and W. Zhao. (1995, November). “Hard Real-Time Communications with Weighted Round Robin Service in ATM Local Area Networks” in *IEEE Proceedings of the 1st International Conference on Engineering of Complex Computer Systems*, pp. 96-103.

[Rey89] J. Reynolds. (1989, December). “The Helminthiasis of the Internet”. RFC 1135. IETF Network Working Group. [Online]. Available: http://www.ietf.org/rfc/rfc1135.txt

[Rey94] J.Reynolds, and J. Postel. (1994, October). “Assigned Numbers”. RFC 1700. IETF Network Working Group. [Online]. Available: http://www.ietf.org/rfc/rfc1700.txt

[Sch00] D. Schnackenberg, K. Djahandari, and D. Sterne. (2000, January 25-27). “Infrastructure for Intrusion Detection and Response”, in *Proceedings of the DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00*. Vol 2, pp. 3-11.

[Sch01] D. Schnackenberg, H. Holliday, R. Smith, K. Djahandari, and D. Sterne. (2001, June). “Cooperative Intrusion Traceback and Response Architecture (CITRA)”, in *Proceedings of the Second DARPA Information Survivability Conference and Exposition (DISCEX II),* Anheim, California, USA.

[Sta02] S. Staniford, V. Paxson, and N. Weaver. (2002, August). “How to 0wn the Internet in your Spare Time”, in *Proceedings of the 11th USENIX Security Symposium*, pp. 149-167. California, USA.

[Ste01] D. Sterne, K. Djahandari, B. Wilson, B. Babson, D. Schnackenberg, H. Holliday, and T. Reid. (2001, September 27). “Autonomic Response to Distributed Denial of Service Attacks”, in *Proceedings of Recent Advances in Intrusion Detection, 4th International Symposium*, pp. 134-139. Davis, California, USA. [Online]. Available: http://link.springer.de/link/service/series/0558/bibs/2212/22120134.htm

[Wan01] Zheng Wang. (March 2001). “Internet Quality of Service: Architectures and Mechanisms”. Morgan Kaufmann Publishers Inc, ISBN: 1-55860-608-4. San Fransisco, California, USA.

[Wei02] N. Weiler. (2002 June 10-12). “Honeypots for distributed denial-of-service attacks”, in *Proceedings of the Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2002. (WET ICE 2002),* pp 109-114.

[Wik03-1] Wikipedia, the free encyclopedia. “Cracker (computing)”. [Online]. Available: http://en2.wikipedia.org/wiki/Hacker

[Wik03-2] Wikipedia, the free encyclopedia. “Hacker”. [Online]. Available: http://en2.wikipedia.org/wiki/Cracker_(computing)