# Ensuring QoS During Bandwidth DDoS Attacks

Moti Geva

Department of Computer Science

Ph.D. Thesis

Submitted to the Senate of Bar-Ilan University

Ramat-Gan, Israel April 2013

This work was carried out under the supervision of Prof. Amir Herzberg, Department of Computer Science, Bar-Ilan University.

# Preface

As a kid, I got the wonderful opportunity to see the digital revolution with my own eyes, and the evolution of the Internet. I recall getting my first computer which had no modem, let alone a network interface card. Calling up friends to come over and play on the computer would usually result in the group coming over with a large box full of floppy disks. My next computer had a fast 56 Kbps modem, which I used for dialing-up and connecting to the Internet. I became fascinated with computers and networking, so it came as no surprise to anybody when I decided to get a degree in computer engineering.

A little over four years ago, I decided to pursue a Ph.D in computer science. At the time, I had already worked for several years in systems R&D, and I had gained practical experience in networking and security. After my M.Sc. thesis got published, a Ph.D. seemed like something that I would enjoy working toward, which turned out to be just right. However, I found out that a Ph.D. is not a list of scientific results, but rather a long, interesting, and sometimes difficult journey.

In research, one knows where he starts but can never know which twists and turns will the research take. My original research proposal was named "DOT-COM: Secure Decentralized Online Trading and Commerce." Over the years the research focus has shifted from the "decentralized online trading" part to the "secure" part, and some results got published [32, 33, 35]. Eventually, the thesis was renamed to "Ensuring QoS During Bandwidth DDoS Attacks," as most of the work was done in that area. Nevertheless, we got to publish a paper on DOT-COM [34], and online trading remains an important motivation for the rest of the thesis.

My Ph.D. goal could not have been accomplished without the help of many people. First and foremost, I had the privilege to have Prof. Amir Herzberg of the Department of Computer Science at Bar-Ilan University (BIU) as my advisor. In addition to being an inspiring researcher, teacher, and advisor, Amir has also been an extremely pleasant person to work with. At the time I started my Ph.D., I couldn't imagine how much I would learn from Amir. During the countless hours we spent discussing the research, Amir was insightful and with keen observations that shined a spotlight on parts in need of further research, refinement and clarifications. Amir taught me so much about research and I am grateful for his guidance. I would especially like to thank Amir for always being available; many times I found myself writing him emails in the middle of the night to find out early in the morning that I had already got a response.

I would like to thank my co-authors. To Yoshi Gev, for his part in the Backward Traffic Throttling research. To Dr. Yair Wiseman who was my M.Sc. advisor and introduced me to the wonderful world of academic research, with whom I published my first research papers. To Avshalom Elmalech, Prof. Barbara Grosz, and Dr. David Sarne, with whom I had the opportunity to work and co-author a paper in the field of artificial intelligence.

For their highly appreciated help, advice and encouragement throughout my Ph.D. process, special thanks goes to: Prof Amihood Amir – former Chair of the Faculty of Exact Sciences and former Chair of the Department Computer Science, Prof. Tomy Klein – Chair of the Computer Science Department, and Prof. Moshe Lewenstein – former Chair of the Computer Science Department. All of whom helped me at various crossroads, from undertaking the Ph.D. to graduating. I am also grateful to Dr. Ariel Frank, Dr. Avinatan Hassidim, Prof. Yehuda Lindell, Prof. Benny Pinkas, and Prof. Ely Porat, for their helpful advice and discussions.

This long period would certainly not have been the same without the pleasant company of other students and friends: Gilad Asharov, Aharon Brodie, Boaz Catane, Ran Cohen, Yair Dombb, Eran Omri, Nethanel Gelernether, Yossi Gilad, Rachel Ginzberg, Haya Shulman, Assaf Tabach, Erez Waisbard, and Hila Zarosim. I would also like to

thank the many students whom I taught throughout the years, and from which I learned a lot. They have made my teaching experience enlightening, fun, and rewarding.

I would like to convey special thanks to my dear friend Dr. Omid David for his lasting friendship, support, stimulating discussions, and helpful suggestions, both professionally and personally.

Most of all, I would like to deeply thank my beloved family. To my loving brother and sisters, Menachem, Michal, and Odeyah-Sarah, for their support and encouragement. To my wonderful children Yoav and Noga, who constantly remind me of what is really important in life.

Finally, to my dear mother Rachel who ever since I can remember myself has always been there for me, encouraging, and doing everything she possibly can for me; and to my amazing wife Merav for her understanding, unwavering support, constant encouragement, and endless love. To these two extraordinary wonderful women, to whom I owe so much, I dedicate this thesis!

*Ramat-Gan,*

*April, 2013.*

# Contents

# List of Tables

# List of Figures

# Abstract

Distributed Denial of Service (DDoS) attacks are aimed at exhausting various resources of victim hosts, thereby preventing legitimate usage of their computational capabilities. DDoS attacks are often launched by organized crime, hacktivists, or other (un)usual suspects, making this type of cyber crime a major concern for many organizations around the world.

The Internet is a best-effort packet-switching network. Everyday usage shows that the Internet is able to properly work, and successfully deliver information across the globe in an instant. However, bandwidth distributed denial of service (BW-DDoS) attacks bring to light the limitations of best-effort networks.

In this thesis we present our research about mitigation of Bandwidth DDoS (BW-DDoS) attacks. BW-DDoS is aimed at exhausting network resources, commonly routers' queue space, and prevent access to the victim server. BW-DDoS attacks have a devastating effect over protocols employing congestion control, as their performance is sharply degraded as a result of losses and delays. BW-DDoS mitigation techniques introduced in this work refrain from making changes to Internet infrastructure equipment, i.e. routers, hence they are focused on adjusting end-host behavior, and the configuration of routers.

The first chapter serves as an introduction to this work. In it we overview key BW-DDoS attacks and defenses. We argue that so far, BW-DDoS has employed relatively crude, inefficient, "brute force" mechanisms; future attacks may be significantly more effective, and hence much more harmful. We discuss current deployed and proposed defenses. We argue that to meet the increasing threats, more advanced defenses should

be deployed. This may involve some proposed mechanisms (not yet deployed), as well as new approaches.

The second chapter details QoSoDoS, a novel protocol that ensures QoS (that is timely delivery) over DoS-prone (best-effort) networks. QoSoDoS is based on scheduling multiple transmissions of packets while attempting to minimize overhead and load, and avoiding self-creation of DoS. On the downside, QoSoDoS retransmission model achieves throughput proportional to the delivery probability, hence for large bandwidth attacks QoSoDoS is able to assure only a low throughput.

The third chapter investigates the effectiveness of using overlay networks to break through congested networks. We present a novel design which, upon congestion, turns to an end-host based overlay to redirect communication and amplify legitimate traffic, using one or multiple paths to the destination. Our overlay is carefully controlled, thus preventing self-generated DDoS. We suggest two ways to construct such an overlay, one way is by using the server's clients and the other way is by using cloud resources. Cloud resources come at a cost which may be substantial, depending on the configuration (e.g. 0.77 cents for 1 megabit of data with packet delivery probability of 1%; see Section 3.2.2 for details), and therefore may be inappropriate for many application or undesired by many users. Additionally, the destination must be instrumented with ways to securely differentiate between legitimate and attack traffic. To that end we propose two MAC-based protocols to enable such differentiation.

The forth chapter discusses Backward Traffic Throttling (BTT), an efficient, decentralized mechanism for congestion and bandwidth-flooding attacks mitigation. BTT employs three basic mechanisms to throttle excessive traffic, namely: prioritizing legitimate flows, shaping traffic, and by requesting upstream BTT nodes to similarly prioritize and shape traffic. BTT is relatively easy to deploy: it requires no changes to routers, and does not modify traffic. Instead, BTT configures routers' queuing discipline and traffic shapers. The broader BTT's deployment is the better its performance. However, BTT has two main limitations. The first limitation is based on the fact that as for any changes

to router, even configuration changes may prove challenging to implement in the real world. The second limitation is that BTT assumes the existence of *typical traffic*. During an attack BTT uses the typical traffic estimation to prioritize legitimate traffic over attack traffic.

# Chapter 1

# Introduction

Distributed denial of service (DDoS) attacks pose a serious threat to the Internet. We discuss the Internet's vulnerability to *Bandwidth Distributed Denial of Service (BW-DDoS)* attacks, where many hosts send a huge number of packets exceeding network capacity, causing congestion and losses, thereby disrupting legitimate traffic. TCP and other protocols employ congestion control mechanisms that respond to losses and delays by reducing network usage. Hence, their performance may be degraded sharply due to such attacks. Attackers may disrupt connectivity to servers, networks, autonomous systems, or whole countries or regions; such attacks have already been launched in several conflicts.

In this chapter we overview the problem of BW-DDoS attacks and defenses. We argue that so far, BW-DDoS attacks employed relatively crude, inefficient, "brute force" mechanisms; future attacks may be significantly more effective, and hence much more harmful. We discuss current deployed and proposed defenses. We argue that to meet the increasing threats, more advanced defenses should be deployed. This may involve some proposed mechanisms (not yet deployed), as well as new approaches, which we will present in later chapters.

## 1.1 Problem Overview

Internet services are indispensable – and yet vulnerable to *Denial of Service* (DoS) attacks, and especially to *Distributed DoS* (DDoS) attacks. In this chapter we focus on DDoS attacks, in which many attacking agents cooperate to cause excessive load to a victim host, service, or network. DDoS attacks have increased in importance, number and strength over the years, becoming a major problem. In a recent survey of network operators [12] (2012, Fig. 10), DDoS was the most common identified "significant threat" (76% of respondents). Furthermore, significant growth in size of attacks (in bytes and packets) and in their sophistication was also reported [12, 41].

We further focus on *Bandwidth Distributed Denial of Service* (BW-DDoS) attacks, which disrupt the operation of the network infrastructure by causing *congestion*, i.e., an excessive amount of traffic. Congestion may be due to the total amount of traffic (in bytes), or the total amount of packets (often a lower limit, using short packets). BW-DDoS attacks can cause loss or severe degradation of connectivity between the Internet and victim network(s) or even whole autonomous system(s), possibly disconnecting whole regions of the Internet. Recently reported BW-DDoS reached volume of 100 Gbps [12] and the latest claimed largest attack reported an attack of 300 Gbps [23]; 60-86.5% of the BW-DDoS targeted the infrastructure (layer 3) [41], including the mitigation infrastructure itself [12].

In this chapter we do not discuss DoS/DDoS attacks which are not focused on bandwidth, including exploits of vulnerabilities and limitations of specific application/transport protocol implementations (e.g., SYN flooding), service (e.g., ping of death) or device (e.g., attack exploiting router and firewall vulnerabilities). For a general DDoS attacks and defenses taxonomy (not limited to BW-DDoS) see Mirkovic et al. [59].

BW-DDoS attackers may use different techniques as well as different attacking agent capabilities. A strong attacking agent is a *privileged-zombie*, i.e., a software agent having high privileges and complete control over the machine on top of which it is being

executed, with the ability to make manipulations to the protocol stack, e.g., being able to send spoofed IP packets. Weak agents include *puppets* – that is, programs downloaded automatically and executed within sandboxes (e.g., scripts and applets), such as JavaScript-based web pages. Next, an attacker may use simple types of bandwidth flooding, or elaborate techniques to amplify bandwidth, such that uncompromised machines assist with consuming bandwidth.

In Section 1.1.1, we discuss significant known BW-DDoS attacks, and compare them with respect to their effectiveness and requirements, considering the attacking agents, protocol manipulations, amplification and attack target. We discuss widely used attacks as per the above mentioned reports [12, 41], as well as more advanced attacks, which so far have only been presented academically. We argue that as more advanced attacks are adopted and cyberwarfare threats increase, BW-DDoS growth rates may further accelerate.

In Section 1.1.2, we discuss important results from the vast body of research regarding BW-DDoS defenses developed in the past twenty years. Some of these defense techniques are widely deployed defenses in practice, and other defenses were only proposed academically. We discuss defenses from both groups, and compare them with respect to the action they take to mitigate the attack, the network location in which they are deployed, the infrastructure adaptations they require, and their dependency on cooperation. Considering reported attacks, we argue that existing defenses may not suffice in the future. Hence, further research, standardization and development of new practical defense mechanisms is required to ensure that defenses can withstand potential increasingly strong attacks.

### 1.1.1 Bandwidth Denial of Service Attacks

BW-DDoS attacks are usually generated from a large number of compromised computers (zombies or puppets). According to recent surveys, e.g., [12, 41], Bandwidth Distributed Denial of Service is the most frequently used DoS method. Most BW-DDoS attacks use a few simple ideas, mainly, *flooding*, i.e., many agents sending packets at the maximal rate,

and *reflection*, i.e., sending requests to a server with fake (spoofed) sender IP address, resulting in the server sending (possibly longer) packets to the victim.

Flooding attacks have created significant damages, since the attackers used a sufficient number of agents to cause massive bandwidth consumption and packet losses. For example, the largest attacks reported in recent years consisting of 100 Gbps (2010) and 60 Gbps (2011, 2012) [12]. Moreover, it seems that gradually, attackers are adopting more complex and effective attacks. Attacks in 2010-2011 were DNS reflection/amplification attacks, described below. In 2012 the attack was aimed at the DNS infrastructure itself. This trend of using more effective attacks is alarming, since significantly more effective BW-DDoS techniques were discovered by researchers (and more may exist).

Nevertheless, inducing a significant percent of packet loss is no easy task for an attacker. Generally, packet delivery probability is the ratio between the available bottleneck link bandwidth and the attack's rate, denoted $P_D$, is independent and can be approximated by:

$$P_D \approx \min\left(1,\ \frac{R_O}{R_I}\right) \tag{1.1}$$

where $R_O$ is the outgoing rate, or link's bandwidth, and $R_I$ is the incoming rate; see Figure 1.1 and Section 2.2.2. However, as depicted in Figure 1.1, due to TCP's congestion control, congestion or (small) packet loss probability causes dramatic performance degradation in TCP connections. It follows that BW-DDoS damage may be worse than mere consumed bandwidth.

Figure 1.2 depicts the results of an Internet scale simulation we have conducted, which can be used to emphasize the potential damage of various-sized BW-DDoS attacks. Details regarding the simulation are discussed in Section 3.4. As depicted in Figure 1.2, large-scale attacks, within the order of magnitude seen to date, can cripple even large ASes, let alone specific hosts or networks.

BW-DDoS attacks come in many flavors, and utilize various mechanisms to induce excessive bandwidth consumption. We will discuss the main features which can be used

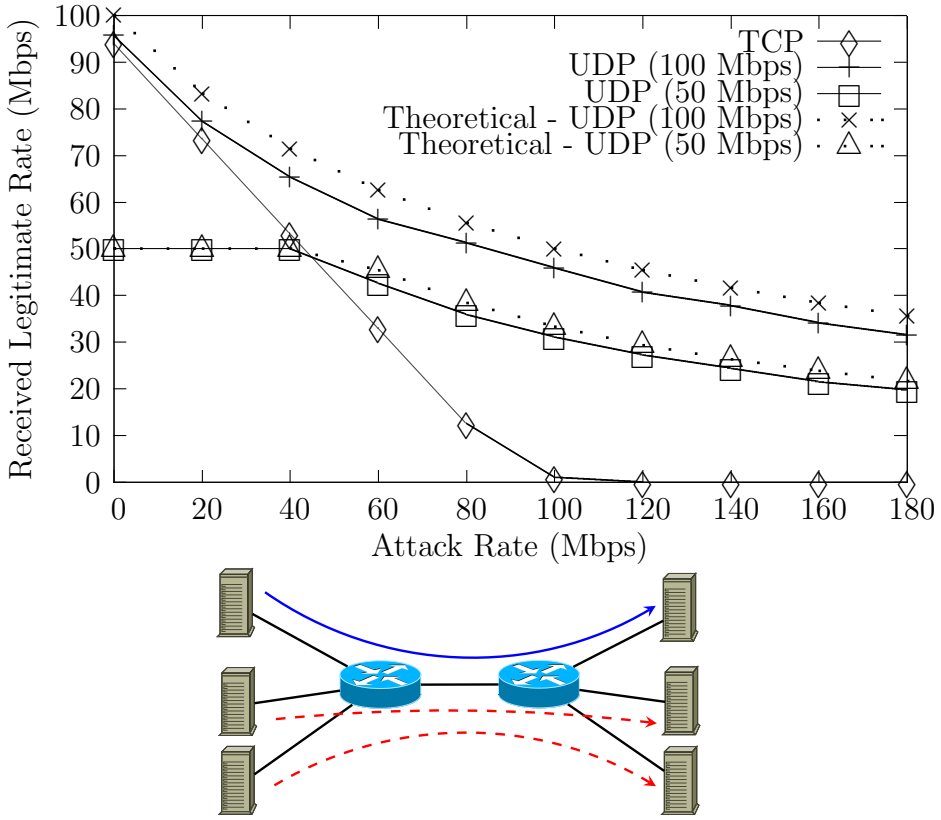



Figure 1.1: Experimental results of delivered rate vs. congestion over a bottleneck link. The diamonds (◊) line is TCP, the pluses (+) and squares (□) are constant rate UDP, and the dashed lines are theoretical rates for constant rate UDP flows as per Eq. 1.1. TCP reduces its rate as a function of available bandwidth, whereas UDP suffers from packet loss. In the topology: the top solid line (blue) represents the legitimate traffic, and the bottom dashed line (red) represents the attacker. All links are 100 Mbps.

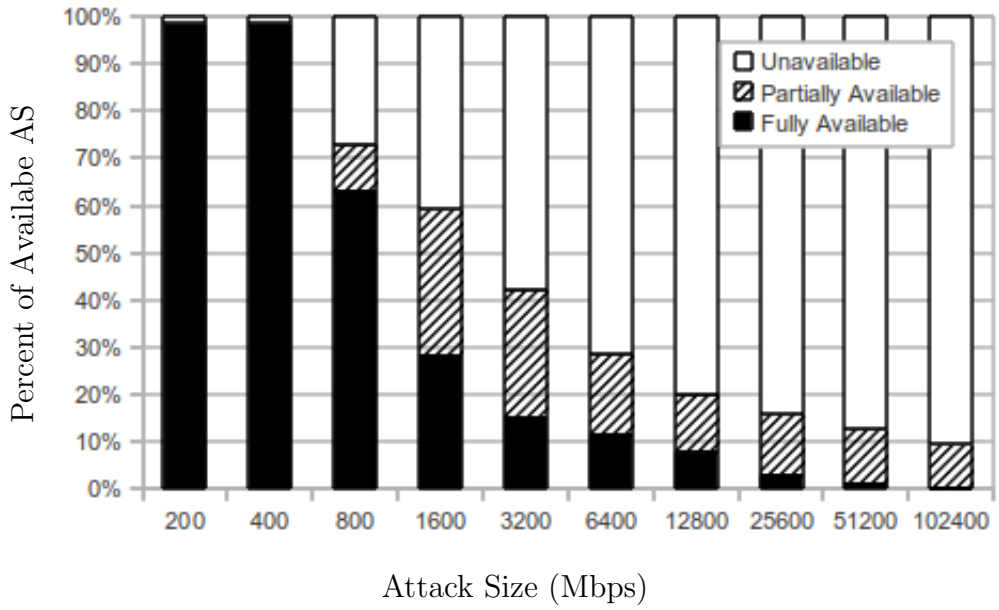



Figure 1.2: Percent of available Autonomous Systems (AS) vs. attack size (Mbps). Black bars are ASes with available bandwidth to support TCP connections from all incoming links. The gray bars are ASes with available bandwidth for only part of their incoming links. The white bars are fully congested ASes which cannot sustain TCP. The figure implies that most of the Internet is prone to DDoS at the attack scales seen to date. The graph is based on the Internet's AS topology simulation, as described in Section 3.4.2.

to classify different attacks, and discuss which capabilities are required from an attacker to launch such attacks.

### Attacking Agent

Let us first consider a naive BW-DDoS attack, where the attacker sends as many packets as possible directly to the victim, or from attacker controlled machines called "zombies" or "bots." The simplest scenario is one in which the attacker is sending multiple packets using a connectionless protocol such as UDP. In UDP flood attacks, the attacker commonly has a user-mode executable on the zombie machine which opens standard UDP sockets and sends many UDP packets towards the victim.

For UDP floods, and many other BW-DDoS attacks, the attacking agents must have *zombies*, i.e. hosts running adversary-controlled malware, allowing the malware to use standard TCP/IP sockets. Other attacks require only *puppets* [11], i.e. scripts, applets, etc., downloaded and run automatically by client agents such as web browsers. Being

untrusted, puppets' operations are restricted by a *sandbox*; e.g., they cannot send UDP packets, let alone spoof packets, and they are limited in establishing TCP connections. Nevertheless, even though puppets cannot induce as much bandwidth as zombies, they can still induce significant bandwidth usage. For example the maxSYN attack [11] aims to maximize SYN packets by setting the sources of several JavaScript image objects to be nonexistent URLs repeatedly every 50 milliseconds. Every time the script writes into the image source URL variable, old connections are stalled by the browser and new connections are established to fetch the newly set image URL.

On the other hand, other types of attacks require zombies to have administrative privileges for execution. We refer to privileged zombies as *root-zombies*. For example, to send packets with a *spoofed* source IP address, a zombie commonly requires the ability to open raw sockets, which is permitted for privileged users only. It is also important to note that some protocol manipulations require network support, or more accurately, lack of prevention. Specifically, spoofing is commonly filtered by ingress filtering as discussed in Section 1.1.2.

Hence, we consider three types of zombies: *puppets, zombies,* and *root-zombies*. Taking control over puppets is relatively easy, and can be done by fooling users to browse to the attacker's website. Zombies are harder to take over, as they require the attacker to install malware on the zombie machine by exploiting some vulnerability or tricking users into installing the malware themselves. Root-zombies require that either an installed zombie was initially installed with high privileges, or that they have a privilege escalation exploit that can gain such high privileges.

#### Amplification Attacks

In a naive attack the attack traffic is limited by the bandwidth capacity of the compromised machines, and the entire load on the victim is due to packets sent by the zombies. Other attacks use the attacking agents' bandwidth more effectively, such that on average, every packet sent by a zombie causes transmission of multiple and/or larger packets to

the victim by non-compromised machines. Specifically, the attacker sends a request $r$ of size $|r|$ which results in longer response $r'$ of size $|r'|$, achieving amplification factor of $\frac{|r'|}{|r|}$.

*DNS amplification* attacks [77] rely on the fact that DNS responses may be larger in size than DNS requests. DNS requests are pretty short, e.g. 40 bytes, whilst responses may be much longer. Originally, DNS responses over UDP were limited to 512 bytes; however, DNS extensions (EDNS) [78] allow longer responses, e.g., 1,500 and even 4,000 bytes. Hence, DNS amplification factor can be $\frac{512}{40} = 12.8$, and even up to $\frac{4000}{40} = 100$ with EDNS. With the uptake of DNSSEC [13], which relies on the long packet capabilities of EDNS, long responses are likely to become more common.

Theoretically, based on an amplification factor of 100 an attacker requires roughly 100 zombies, each sending DNS requests at 100Kbps, to achieve a 1Gbps attack. For a 10Gbps attack 1,000 zombies are required, etc. Alarmingly, significantly larger botnets, consisting of hundreds of thousands of zombies, have already been discovered.

### Protocol Manipulations

We discuss two types of protocol manipulations which can be used by an attacker. The first attempts to avoid detection, and the second tries to exploit legitimate protocol behavior and cause legitimate clients/servers to excessively misuse their bandwidth against the attacked victim. Typically, protocol manipulation for bandwidth attacks require a strong zombie with administrative privileges, as the manipulation is commonly done at the low protocol layers handled by the operating system, usually IP and TCP.

For naive attacks such as UDP floods, the sources of the attack are visible. The victim host can see the zombies' source addresses, making it relatively easy to block packets and to take technical or legal measures against the attacking machines and their owners. Therefore, an attacker may try to *avoid detection* by manipulating its source IP address, commonly referred to as *spoofing*. Thus an attacker can send multiple packets, each containing a different spoofed source address, making it harder for the victim to

identify the attacker source, and block them.

The second type of attack exploits legitimate protocol behavior by misusing packet fields, or acting dishonestly, causing a legitimate host to send unwanted or excessive packets to the victim. For example, using IP spoofing an attacker can induce a DNS reflection attack. DNS is a request-response protocol; that is, a DNS server or resolver, will return responses to clients that issue DNS requests. The client return address is determined using the source IP address appearing in the request. DNS reflection attacks exploit this behavior: the attacker sends a spoofed DNS request to an *open resolver* [2, 28], making the open resolver issue a response packet to the spoofed address. A DNS resolver is *open* if it provides recursive name resolution for clients outside of its administrative domain. Commonly, the attacker uses the victim's IP address as the spoofed source address. Further, the attacker commonly amplifies his attack, causing the reflected response to be longer, as described above.

Some attacks, such as *optimistic-acknowledge* (opt-ack) [70], take advantage of other mechanisms, such as TCP congestion control. Generally, the congestion control mechanism adapts TCP transmission rate based on available bandwidth. Typically, the assumption behind the congestion control mechanism is that the (main) reason for packet loss is congestion. Hence, whenever TCP packets are successfully received, the rate increases, and whenever packets are lost the rate is reduced. The way that TCP knows whether a packet has been received or not is based on ACK(nowledge) packets sent by the destination. The opt-ack attack idea is therefore simple: a malicious client sends some request to a server, then, as the server sends the response packets, the client *optimistically acknowledges* receiving them by sending ACK packets, without actually receiving the packets themselves. Thus, very low bandwidth is required to cause servers to send lots of traffic, limited mainly by the servers' bandwidth.

An *ACK-Storm* [4] attacks TCP's acknowledge mechanism. The attacker eavesdrops on an existing TCP connection. Next, the attacker spoofs ACK packets with a higher sequence number than what was actually sent; this induces an ACK packet back containing

| **Attack** | **Agent** | **Amp** | **Proto. Manipulations** | **Target** |
|---|---|---|---|---|
| UDP Flood | zombie | 1 (none) | No | Host |
| maxSyn [11] | puppet | 1 (none) | No | Host |
| DNS Reflection [77] | root-zombie | $< 100$ | Yes (IP spoof) | Host |
| opt-ack [70] | root-zombie | $> 1,600$ (see [70]) | Yes (TCP) | Host |
| ACK Storm [4] | root-zombie | $> 40,000$ (see [4]) | Yes (IP spoof) | Host |
| Coremelt [73] | zombie | 1 (none) | No | Link |

Table 1.1: Comparison Between BW-DDoS Attacks

the real sequence number. Sending such packets to both connection ends, respectively, induces a repeated back and forth exchange of ACK packets, until either end eventually terminates the connection.

### Attack Target

While most BW-DDoS target specific hosts or networks, new types are also trying to attack target links. For example, Coremelt [73] uses a peer-to-peer model in which zombies communicate directly with each other. Amongst $N$ zombies there exist $O(N^2)$ routes, some of which use the victim (core) link. The attacker can then create excessive traffic on the link using only inter-zombie communications. Coremelt requires regular zombies without high privileges, as it can use the standard TCP/IP stack without any protocol manipulation. Coremelt type of attacks are mainly theoretical since the Internet backbone links are highly provisioned and would require huge peer-to-peer networks to clog. Moreover, based on CAIDA data sets [3], we can say that there are more than a 100,000 links between more than 35,000 different autonomous systems, making it very hard to take down almost any specific link. Nevertheless, assuming enough zombies can be obtained, Coremelt may prove difficult to detect and filter, since each connection can use a small amount of bandwidth.

Concluding this section, we argue that deployed attacks use relatively crude methods,

while future attackers will likely be able to use significantly more effective attacks. It is very hard to determine how difficult it is for an attacker to actually launch such advanced attacks in the Internet, but the basic know-how is being studied and to some extent even demonstrated. Nevertheless, existing attacks and especially advanced attacks may challenge currently deployed defense mechanisms, motivating investigation of new mechanisms.

### 1.1.2 Network-level Defense Mechanisms

BW-DDoS defense mechanisms focus on several types of schemes, including detecting, filtering, absorbing, cooperating and coding. In this section we survey defense schemes of both deployed and academically proposed mechanisms, and provide examples for the various schemes. Note that many, but not all, defense mechanisms rely on the ability to differentiate between attacks and legitimate flows. For example, Jin et al. [44] proposed a spoofing filter based on hop-counts differentiation between spoofed and real packets. Ingress filtering [29] aims to block spoofed packets at the source ISP, by identifying that the spoofed packet cannot have originated from that ISP. LOT [36] tunnels packets between two hosts using a nonce, preventing a non-MITM attacker from spoofing packets between these hosts. Carl et al. [20] present a survey regarding various detection techniques.

#### Defense Mechanism Location

There are various defense mechanisms which can be deployed at different network locations. A defense mechanism can be deployed close to the destination by the victim. Note that defense mechanisms close to the destination may get a good idea about some attacks' properties, but for mitigation of BW-DDoS they might not be well positioned, since many packets are already discarded near the victim. Hence, many defense mechanisms try to mitigate the attack closer to its sources.

Router or backbone-based defense mechanisms are usually located near an over-

provisioned link, and try to make sure that the traffic reaching the victim will be mostly from legitimate sources rather than attack flows. Similarly, source-based defense mechanisms, such as D-WARD [61], try to prevent an attacker from sending excessive traffic, especially during a BW-DDoS.

Additional network locations may be *in the cloud* or *overlays networks* – that is, traffic is routed via an over-provisioned cloud service which "scrubs" the attacking flows and forwards only legitimate traffic to the victim.

### Infrastructure Adaptations

A concern that may affect the ability to deploy BW-DDoS solutions is the amount of changes that the infrastructure has to undergo. For example, some solutions require installing new software at end-hosts, some require software updates to routers, while other solutions may be satisfied with mere configuration of networking equipment.

There are several reasons to refrain from deploying BW-DDoS mitigation solutions. One reason is that it is very difficult to estimate the impact of deploying different solutions, and considering the network equipment itself – "if it ain't broke, don't fix it." However, we can assume that some changes are easier to make than others. For example, configuration changes should be significantly easier to make than most other changes, as the existing software and hardware are used. Next, software changes at end-hosts are assumed to be more difficult than configuration. Finally, we assume that any change to routers (that is, software/firmware and especially hardware) are very difficult to make and widely deploy.

### Filtering Schemes

Assuming the offending flows are identified, they can be filtered out, or blocked. Filtering can take place in various network locations: close to destination, at the core (i.e., routers), or close to source. Usually, to be effective in BW-DDoS mitigation, filtering should occur before the congested link, as the victim itself is usually not in position to hold the attack

back.

One filtering example is preventing source IP spoofing. RFCs 2827 [29] and 3704 [15] (BCPs 38, 84), recommend that ISPs employ *ingress filtering* and filter packets with non-permitted IP addresses. This is performed by many ISPs, however, the Spoofer Project [5] and Arbor reports [12] indicate that approximately 15% of Internet addresses can still send spoofed packets.

Additional deployed filtering mechanisms include *Access Control Lists* (ACL), *Remote-Triggered Black Hole* (RTBH), and firewalls. ACLs are router mechanisms which allow or deny matching flows. ACLs are often configured manually, however, some Intrusion Prevention Systems (IPS) can configure ACLs automatically. Each ACL entry takes a significant amount of memory and some time to process, therefore a router should limit the ACL rules it holds, both in number as well as the processing time they take. Both memory and CPU usage increase as more ACL entries are used, which may be an additional cause for DDoS – not necessarily bandwidth based.

RTBH (RFC5635) [47] uses the router's forwarding tables, such that *all traffic* to the victim, or from attacking sources, is forwarded to a "black-hole," completely denying access to the target. Forwarding-rules processing is commonly faster than ACL processing, however, RTBH filtering is significantly more aggressive, and may help an attacker to disconnect its victim from its sources/destination, thereby potentially achieving the attacker's goal, with little resources.

*dFence* [56] is a transparent mechanism to the existing Internet infrastructure with no software modifications at either routers or the-end hosts. dFence dynamically introduces special-purpose middlebox devices into the data paths of the hosts under attack, which filter offending flows.

### Rate-Limiting Schemes

In contrast to completely blocking the attacking flows, *rate-limiting* schemes let the offending flows transmit their *typical rate*, or obey some other limit.

*Rate-limiting* at routers was proposed in the literature in several main forms, *capabilities*, *packet tagging*, and *scheduling-based*. *Capabilities* are tokens issued by the destination (server) to the source (client), and informs that the destination is willing to accept traffic from the source. The issued capability is attached to packets sent by the source, allowing routers en route to identify and prioritize approved flows.

*SIFF* [83] proposed stateless capabilities, in which the capability is calculated using (keyed) hash. Routers check and prioritize flows carrying verified capabilities. *TVA* [85] keeps a (small) state in routers, and allows servers to request specific restrictions per flow. Capabilities-based solutions assume that the victim will only authorize legitimate sources, and not cooperate with the attacker. Deployment of capabilities-based solutions will require change to both end-hosts and routers.

*PSP* [22] collects network statistics at the provider level, and infer the *typical traffic rates* between origin–destination pairs. Packets are *tagged* upon arrival to the provider as either normal or excessive. Whenever a router gets congested, packets tagged *excessive* are discarded first, effectively prioritizing packets tagged as normal. PSP deployment is only within the provider boundaries, and requires changing routers' software for the packet tagging and prioritizing, or otherwise taking advantage of existing IP packet fields, which may be used by different applications and hence potentially cause damage to some flows. *D-WARD* [61] does not tag excessive packets; instead it simply discards of them at the source-end, based on collected statistics.

### Cooperation Schemes

*Pushback* [43] schemes, such as *ACC* [55], *AITF* [14] and *StopIt* [51] are router-based mechanisms trying to prevent DoS attacks by blocking attack streams close to the attack source. This is done by first identifying the attack and propagating a block request upstream towards attack source(s).

In Pushback [43], the victim identifies the attacking flows' profile, followed by "pushing the attack back," and freeing the victim's resources to handle legitimate traffic. *FlowSpec*

(RFC5575) [57], describes an operational implementation similar to key ideas in Pushback. Basically, Pushback and FlowSpec are ACL-like filtering schemes, but instead of having the ACL entries employed within a single AS, they are distributed and pushed back upstream.

Pushback-based solutions allow under-provisioned nodes to filter offensive traffic away from the victim. However, the assumption that victim nodes can identify the attack profile might prove very difficult. Furthermore, like other ACL schemes, many Pushback requests lead to many filtering rules and ACL entries, and may result in a DoS attack on routers' processing capabilities, or enable the launch of a decoy attack to exhaust filtering rules, followed by an attack on the real target. Alternatively, this type of cooperation may let an attacker issue a Pushback request, disconnecting the victim.

*DefCOM* [63] introduces a core framework which enables cooperation between source, core and destination defenses during an attack, by using both shaping and Pushback schemes.

Cooperation-based schemes assume that the cooperating nodes are honest with each other in the sense that they will only propagate upstream requests upon BW-DDoS attacks, which is something that is debatable. Alternatively, the signaling plane between cooperating nodes may by itself be a target for an attack.

### Overlay and Cloud Schemes

*Overlay networks*, i.e., the use of helper nodes as intermediaries between source and destination to ensure availability even when under severe DDoS attacks. Overlay networks research can generally be divided into two types: *Indirect Overlay Networks* (ION) and *absorption overlays*.

*IONs*, such as *Resilient Overlay Networks* [9], *OverQoS* [74], and *Bandwidth-Aware Routing in Overlay Networks* [50], bypass the network end-to-end routing, overcoming BGP's shortcomings, such as update speed, route selection under different matrices, or utilizing special network features such as multihoming; see MONET [10]. IONs can

implicitly mitigate BW-DDoS only when some routes are congested while others are not, as depicted by the gray bars of Figure 1.2.

*Absorption overlays* are over-provisioned with bandwidth, and are able to absorb the BW-DDoS. Absorption overlays commonly construct a perimeter around the victim server, which can be penetrated only by selected nodes. The overlay nodes are responsible for authenticating the client, possibly requiring proof of work etc., and forwarding packets to one of the authorized selected nodes to forward traffic to the server; unauthorized traffic is filtered.

"In the cloud" (practical) or "overlay" (academic) solutions, route traffic via the cloud/overlay, which "scrubs" the attack flows. Absorption overlays/clouds are specifically designed to mitigate BW-DDoS, and were investigated in several works, such as *SOS* [45], *Mayday* [25] and *Phalanx* [27], and may also incorporate additional schemes. Note that usually, overlay solutions introduce new protocols, and hence typically require updating host software. Other solutions, mainly deployed solutions, make no protocol changes, and instead rely on configuring BGP or DNS records as to divert the traffic to a cloud-based scrubbing service.

### Coding Techniques

Coding techniques, especially Forward Error Correction (FEC) codes, can be considered for handling various packet loss scenarios. FECs adding redundancy at the bit level are irrelevant for general use in IP networks in which the infrastructure delivers packets, not bit streams.

Inter-packet FECs, such as fountain codes, can be considered to address the problem presented in this chapter. Fountain codes, or rateless erasure codes, such as LT codes [54] and Raptor codes [52,53,71], are forward error correction (FEC) codes, which can generate from the source message as many encoding symbols as needed. The receiver should be able to decode the source message in very high probability from almost any set of encoding symbols of sufficient cardinality – which is about the number of source symbols or very

slightly more.

Fountain coding solutions or fountain-based protocol (FBP), such as presented in papers by Bonald et al. [17] and Raghavan et al. [66], propose that clients transmit at their maximal rate, and overcome losses at the receiver side by using fountain codes. Such methods, if deployed in the current Internet, will severely harm the current TCP-friendly protocols, as well as protocols relying on high delivery probability such as voice and video applications; hence, such solutions cannot be gradually deployed. Other schemes try to cope with losses unrelated to congestion [75], such as wireless networks, and instead of using ACKs to detect congestion losses, they use RTT changes as a congestion indicator, such as in TCP Vegas [19]. Note that upon high congestion the throughput should be decreased, as in other TCP-friendly solutions.

Another important issue is that for small messages, containing only a single or just a few packets, such as short DNS or HTTP requests, fountain codes are inappropriate. Hence, for solutions which try to maintain current TCP-friendliness, and/or handle transmissions of messages with very few packets, fountain codes may be inappropriate altogether.

## 1.2 This Thesis

In this thesis we present novel schemes for Bandwidth DDoS (BW-DDoS) mitigation. Our proposed schemes aim to to provide solutions which can be deployed in today's Internet without having to change any of the Internet's infrastructure equipment software or hardware, especially in regards to routers. We offer three types of solutions: end-to-end, overlay-based, and router-configuration based. These types of solutions are imperative to solve today's problems without redesigning the entire Internet.

Table 1.2 compares between existing solutions and the new solutions proposed in this dissertation. The table compares the different mechanisms based on the *action* they take, where the solution is *located* in the network, what type of *infrastructure adaptations* are

| **Mechanism** | **Action** | **Location** | **Infra. Adapt.** | **Cooperation** |
|---|---|---|---|---|
| Ingress Filter | filter | router | configuration | standalone |
| ACL | filter | router | configuration | standalone |
| RTBH | filter | router | configuration | inter AS (BGP) |
| Capabilities | rate-limiting | dst, router, src | router software | inter AS |
| dFence | filter | core (middlebox) | configuration | intra AS routing |
| D-WARD | rate-limiting | router | software | standalone |
| DefCOM | rate-limiting | router, src, dst | software | dst, router, src |
| PSP | rate-limiting | router | router software/IP fields | intra AS |
| Pushback | filter | router | router software | inter AS |
| RON | detour | src, cloud | end-hosts software, cloud | end-host, overlay |
| SOS | filter | src, cloud | end-hosts software, cloud | end-host, overlay |
| Scrubbing | filter | router, cloud | configuration, cloud | inter AS (BGP), cloud |
| Fountain Codes | coding | src, dst | (all) end-hosts software | end-hosts |
| QoSoDoS | break-through | src, dst | end-hosts | end-hosts |
| Controlled Overlays | break-through | src, dst, cloud | end-hosts, cloud | end-hosts, cloud |
| BTT | rate-limiting | router | configuration | inter-AS (app.) |

Table 1.2: Comparison Between BW-DDoS Defense Mechanisms

required and is the mechanism *cooperative* or standalone. The following chapters present solutions found at three network place: end-hosts (QoSoDoS), overlay/cloud (Controlled Overlays) and routing core (BTT). Neither of the proposed solutions tries to identify or filter the attack. All of the proposed solutions are complementary to existing detection and filtering solutions. QoSoDoS and Controlled Overlays try to break through the congestion which differs from most existing solutions, whereas BTT tries to prioritize the typical traffic and rate-limit the rest of the traffic based on *existing* router mechanisms. All of our proposed solutions try to make changes in places which are relatively easy to change. Deployment difficulty increases alongside the dissertation chapters. End-hosts is probably easier to implement than cloud services and router configuration is probably harder to achieve especially across different ASes. The rest of the chapter shortly describes the different solutions and compares them to prior work.

### 1.2.1 QoS Over DoS-prone Networks

In Chapter 2 we present an end-to-end transport protocol named *QoSoDoS*, that ensures (modest) Quality of Service among peers connected via an unreliable, clog-prone network, which *usually* has much higher bandwidth, such as the Internet. When QoSoDoS is unable to deliver data, it begins sending each packet many times, until the packet is received correctly at the destination. This mechanism assumes that even under BW-DDoS attack, packet delivery probability is larger than 0; we have experimentally validated this assumption. This approach stands in contrast to existing reliable transport protocols such as TCP [64], which responds to packet loss by reducing its rate and even aborting the connection. In essence QoSoDoS overrides TCP's congestion control [8] in cases of extreme congestion, while refraining from self-clogging the network. QoSoDoS limitations are based on its retransmission model. First, the throughput proportional to the delivery probability, hence as the attack increases the lower throughput QoSoDoS can assure. Consequently, for high-bandwidth attacks which cause very-low delivery probability this solution is only applicable to mission-critical low-bandwidth applications. Second, QoSo-

DoS requires that the user estimate what is the worst attack that an attacker can launch and based on that estimation the user configures the worst-case delivery probability.

QoSoDoS is different from the solutions described in Section 1.1.2, as QoSoDoS is an end-to-end solution, requiring no changes to the Internet's infrastructure, and is deployed merely at end-hosts. QoSoDoS is focused on circumventing the congestion control mechanism, hence it can be complementary to many exiting techniques, such as various filtering schemes, coding techniques, etc.

The QoSoDoS retransmission scheme resembles *Application-level* DDoS defense such as *Speak-Up* [79], which requires legitimate sources to "pay" for the service using bandwidth. Speak-Up uses TCP-based traffic as the currency, and makes an auction for each request before passing it to the server. Our basic assumption is that during a bandwidth flooding DoS attack, which occurs at the network level, Speak-Up, as well as other application-level defenses, will not be able to work at all.

### 1.2.2 Controlled Overlays

In Chapter 3 we present a new type of overlay which cooperates to aggregate the available bandwidth and possibly multiple paths to break through congested networks. We present a novel design which, upon congestion, turns to an end-host based overlay to redirect communication and amplify legitimate traffic, using one or multiple paths to the destination as necessary. Our overlay is carefully controlled, thus preventing self-generated DDoS.

Our controlled overlay integrates the two previously proposed schemes for overlays, IONs and absorption overlays, by having multiple overlay nodes, some of which may have better routes to the victim host, along with the capability of absorbing the attack as an absorption overlay.

Absorption overlays, as described in Section 1.1.2, assume that the destination location is well guarded. Clients must relay via the overlay, and congestion-controlled overlay nodes are able to transmit data to the destination. In contrast, we assume that all clients can transmit directly to the destination host, the overlay can assist the client and in-

crease its bandwidth, and we can deliver information even when congestion-controlled protocols fail. Hence, an attacker *flooding the guarded perimeter itself* may cause previously proposed absorption overlays to fail, whereas our overlay should be able to deliver information and utilize the real delivery probability, traverse the congested perimeter, and reach the destination. Note that an attack on the overlay itself is possible for both previously proposed overlays as well as ours. This hazard can be mitigated by using cloud services and instantiating new instances to absorb the attack.

ION schemes, such as *Resilient Overlay Networks* [9], *OverQoS* [74], and *Bandwidth-Aware Routing in Overlay Networks* [50] do not specifically target DDoS but rather improve the QoS of the network. In reference to Figure 1.2, these solutions consider routing improvements for the gray and black bars, whereas for large-scale flooding attacks, the inaccessible ASes (white bars) should also be considered, which is what our solution does.

The controlled overlay has few limitations. The first limitation is that we need to actually construct an overlay network. In Chapter 3 we suggest two possible ways to do that. The first is by using the server's legitimate clients and the second is using cloud resources. To prevent an attacker from impersonating a legitimate node, both client and overlay nodes, we propose two MAC-based authentication protocols which lets the server authenticate the end-client and the overlay node. If either is compromised the server can drop the packets. Note that the solution is inappropriate in setups in which the server cannot distinguish between legitimate and attack traffic. If one decides to use cloud resources instead of using the server's legitimate clients this, one should be aware that these resources have non-negligible costs; we discuss these details in Section 3.2.2. Finally, similarly to QoSoDoS, for very large volume attacks which cause a very low delivery probability this solution is applicable for mission-critical attacks.

### 1.2.3 Backward Traffic Throttling

In Chapter 4 we present Backward Traffic Throttling (BTT), an efficient, decentralized mechanism based on router configuration. BTT employs the following mechanisms to

throttle excessive traffic: prioritizing incoming flows, shaping outgoing traffic, and requesting upstream BTT nodes to similarly prioritize and shape traffic. BTT is relatively easy to deploy: it requires no changes to routers, and does not modify traffic. Instead, BTT configures routers' queuing discipline and traffic shapers. BTT has three main limitations. The first limitation is that any change to routers, even if it is only a configuration change, is non-trivial in a real world production network. The second limitation is that BTT assumes the existence of *typical traffic* between ASes and that the routing between ASes doesn't change dramatically during the attack, that is, changes that are not reflected in the estimated typical traffic change over time. The third limitation is that BTT does not differentiate between attack and flash crowd ("Slashdot effect") and whenever a bottleneck link is congested, BTT kicks into action and prioritizes traffic based on the typical traffic estimations.

BTT is oblivious to the actual attack. Unlike BTT, Pushback schemes rely on identifying specific attack characteristics, such as flow volume, source IP etc. Moreover, Pushback routers require blocking assistance from upstream routers, otherwise the number of blocked flows may be too large to handle; this may also burden deployment and adoption. On the other hand, BTT simply rate limits the traffic and requires few resources from the router. Additionally, spoofing sources introduces difficulties in placing the stream filtering rules at the right place, whereas BTT simply reacts to abnormal traffic rates. Finally, Pushback schemes propose features which commonly require router adaptations, whereas BTT uses existing mechanisms and merely configures them on-the-fly.

Tagging schemes, such as PSP, may be harmful due to usage of non-standard fields which may be problematic in some cases, such as various tunneling and traffic engineering. PSP is based on origin-destination pairs; hence it requires up to (theoretical) $O(\#AS^2)$ rules, whereas BTT requires only $O(\#AS)$ rules (in the worst case) per congested route. Finally, PSP is an intra-AS system and cannot decrease the amount of false positives, whereas BTT, which is an inter-AS system, may improve as its deployment widens.

Yau et al. [86] suggested a throttling (shaping) mechanism with a Pushback mecha-

nism, however, unlike BTT, the solution requires router changes. Throttling is server-centric and is initiated by the destination and not by congested links, and it does not use queuing techniques for packet prioritizing.

Table 1.3 compares BTT and key router-based related works based on *traffic manipulation* (TM), *route modification* (RM), amount of required *router resources* (RR) by the mechanism, *attack aggregate identification* (AAID) and false positives (FP).

Mechanism that use TM are ones that need change the data plane packets, such as packets tagging. Such changes have two main limitations. The first limitation is that it commonly requires router modification. The second limitation is that it may need to use protocol fields which may be used or required by other applications – whether standard or not. RM means that additional functionality has to be added to the router, which implies that the router software or hardware has to be updated. RM is opposed to router configuration which require activating or deactivating an existing router mechanism. Essentially, as any router change is difficult to deploy, requiring a software, let alone hardware, upgrades makes the solution less and less practical in the real world. RR compares the amount of resources the router has to use which may affect the processing, memory and storage requirements. Significant resource usage can possibly slow down the router or even exhaust its resources altogether, potentially becoming the target of DoS attacks. AAID describes whether the mechanism is based on detecting the attacker's flows or not. This is important as differentiating between attack and legitimate traffic is a hard problem and attackers have a tendency to find new ways to bypass identification mechanisms. Finally, FP is the amount of false positive we would expect to see from each mechanism.

| Sys | TM | RM | RR | AAID | FP |
|---|---|---|---|---|---|
| PSP | ✗ (always) | ✗ (yes) | $(\#AS)^2$ (modest) | ✓(no) | yes |
| RTBH | ✓(no) | ✓(no) | $\#AS$ (low) | ✓(no) | high |
| ACL | ✓(no) | ✓(no) | $\#IP$ (high) | ✗(yes) | low |
| FlowSpec/ Pushback | ✓(no) | ✓/✗ | $\#IP$ (high) | ✗(yes) | low |
| BTT | ✓(no) | ✓(no) | $\#AS$ (low) | ✓(no) | deployment dependent |

Table 1.3: Comparison Between BW-DDoS Defense Solutions. TM: Traffic Manipulation (e.g. tagging), RM: Router Modifications (requires non-standard router features), RR: Router Resource (amount of rules/memory), AAID: Attack Aggregate Identification (i.e., requires the identification of attack flows), FP: False Positive.

# Chapter 2

# QoS Over DoS-prone Networks

We present QoSoDoS, a protocol that ensures QoS over a DoS-prone (best-effort) network. QoSoDoS ensures timely delivery of time-sensitive messages over unreliable networks, susceptible to high congestion and bandwidth-flooding DoS attacks. QoSoDoS is based on scheduling multiple transmissions of packets while attempting to minimize overhead and load, and avoiding self-creation of DoS. We present a model and empirical results of QoSoDoS implementation. Our results show that under typical scenarios, QoSoDoS can handle high congestion and DoS attacks quite well.

## 2.1 Introduction

The Internet is a best-effort delivery network – there is no allocation of resources to connections, and hence the Internet provides no guarantee of message delivery, and no bounds on delay. If the network is congested, then routers drop arbitrary packets instead of transmitting them to their destinations. This is often exploited for *Bandwidth Denial-of-Service (BW-DoS)* attacks, in which an attacker intentionally causes excessive traffic and congests the network. In many cases, DoS attacks and especially bandwidth attacks are carried out using multiple zombie hosts, and are referred to as Distributed DoS (DDoS) attacks [60]. Such bandwidth DoS attacks are hard to prevent and mitigate in

an open, best-effort network such as the Internet.

Since the Internet is subject to congestion and bandwidth attacks, critical applications, such as inter-bank clearing, usually resort to private networks, or use *Quality of Service (QoS)* protocols, e.g. Diffserv [37] and MPLS [68], to connect between source and destination. This can be expensive, even for services which can tolerate large (but bounded) delays, if their availability is critical. Examples for such services include emergency services messaging, and financial and other contractual obligations, in which each contract counterparty must stand up to its obligations within a given timeframe, which is usually within the order of seconds, minutes, hours or even days, e.g., DOT-COM [34].

This appears frustrating; the Internet provides low-cost, ubiquitous communication which usually works fine, even for high bandwidth applications such as voice and video calls. Yet, for critical applications such as financial transactions, which require low bandwidth and allow significant delays, we must resort to specialized, expensive infrastructure. *Can we use the unreliable-but-high-bandwidth connectivity of the Internet to provide the reliable-but-modest-bandwidth requirements of financial and other sensitive applications?*

We answer this question in the affirmative, by presenting QoSoDoS, an end-to-end transport protocol that ensures (modest) Quality of Service, among peers connected via an unreliable, BW-DDoS-prone network, which *usually* has much higher bandwidth, such as the Internet. The principle behind QoSoDoS operation is simple: when QoSoDoS detects frequent packet losses, it begins sending each packet many times, until the packet is received correctly by the destination. Assuming a known bound on packet loss probability, and an application-defined target for allowed probability of loss, we can easily calculate the required number of retransmissions. This mechanism assumes that even under bandwidth attack, we can bound the probability of successful transmission; we have experimentally validated this assumption. The QoSoDoS approach stands in contrast to existing reliable transport protocols such as TCP [64], which responds to packet loss by reducing its window (and rate), and even aborting the connection.

QoSoDoS is aimed to override TCP congestion control [8] and maintain crucial com-

munication in cases of extreme congestion. This is complementary to existing research on mitigation of bandwidth and other DoS attacks, which mostly attempts to prevent the BW-DDoS (or other attack), identify the attack and/or its source, and so on; see Chapter 1. QoSoDoS is also complementary to approaches using inter-packet coding, which try to overcome loss which is unrelated to congestion, while still performing congestion control, such as [75]. On the other hand, QoSoDoS does not suggest giving up on congestion control altogether; hence it differs from solutions such as proposed by Bonald et al. [17] and Raghavan et al. [66], which offer replacing congestion control with coding.

An important concern is that QoSoDoS must not clog the network itself by sending more packets than the network can handle, especially as a response to momentary congestion (or attack). Our experiments show that even when many QoSoDoS connections are transmitting over the same bottleneck link, they do not clog the link, and have comparable performance to regular TCP connections. Furthermore, even a large number of QoSoDoS clients recover quickly from short-lived BW-DDoS attacks, with better performance than regular TCP connections.

The rest of the chapter is organized as follows. Section 2.2 describes the QoSoDoS model. Section 2.3 presents the design of QoSoDoS. We describe several transmission schemes in Section 2.4. In section 2.5 we present our experimental evaluation of QoSoDoS. Section 2.6 presents conclusions.

## 2.2 The QoSoDoS Model

Much of the research on networking considers one of two very different models: *QoS (Quality of Service) networks* and *DoS/BE (Denial-of-Service-prone, Best-Effort) networks*. QoS networks are expected to always ensure some maximum delay guarantees, e.g., no losses; DoS/BE networks are usually modeled as completely unreliable, i.e. can drop all packets.

We believe that it is reasonable to regard most DoS/BE (Best-Effort) networks, and

in particular the Internet, as assuring QoS properties, such as bounded delay, but only with very *low* packet delivery probability $P_D$, i.e. $0 \lesssim P_D \ll 1$. We believe that delivery probability $P_D$ holds even when a connection is undergoing a BW-DDoS attack. To support our claim we present a queuing analysis and a discussion in Section 2.2.2, as well as supporting empirical results.

Furthermore, we believe that in reality, packet losses are possible also with QoS networks; of course, in QoS networks, packets are delivered with very high probability $P_Q$. Nevertheless, $P_Q < 1$, namely, there is some probability for some packet loss, e.g., due to electrical interference; hence, $0 \ll P_Q \lesssim 1$. See Table 2.1 for a comparison between DoS/BE and QoSoDoS parameters.

Under benign network conditions we assume that the *effective delivery probability*, denoted $P_E$, is relatively high, that is $P_E \to P_Q$, whereas under (very large) BW-DDoS attack $P_E \to P_D$. Generally $P_D \leq P_E \leq P_Q$.

Throughout this chapter we use subscript $_D$ to describe a *DoS-prone*-related parameter, subscript $_Q$ to describe a *QoS*-related parameter, and subscript $_E$ to denote an *effective* (or actual) value of a parameter. For example, if $P$ denotes the delivery probability, then $P_D$ is the DoS-prone network delivery probability; $P_Q$ is the QoS delivery probability, and $P_E$ is the effective delivery probability of the network. Table 2.3 provides a summary to all notations in this chapter.

To make a DoS/BE network assure packet delivery with high probability ($P_Q$), we may need to retransmit packets. Each time we resend a packet increases the probability for successful delivery. Generally, if a packet is sent $n$ times, and the delivery probability is independent and at least $P_D$, then the probability that at least one copy of the packet is delivered is at least $1-(1-P_D)^n$. Hence, to ensure packet delivery with high probability $P_Q$, we need a large enough $n$ such that $P_Q = 1-(1-P_D)^n$; hence:

$$n \equiv \lceil \log_{(1-P_D)}(1-P_Q) \rceil \tag{2.1}$$

### 2.2.1 Modeling QoSoDoS As Latency-Rate Server

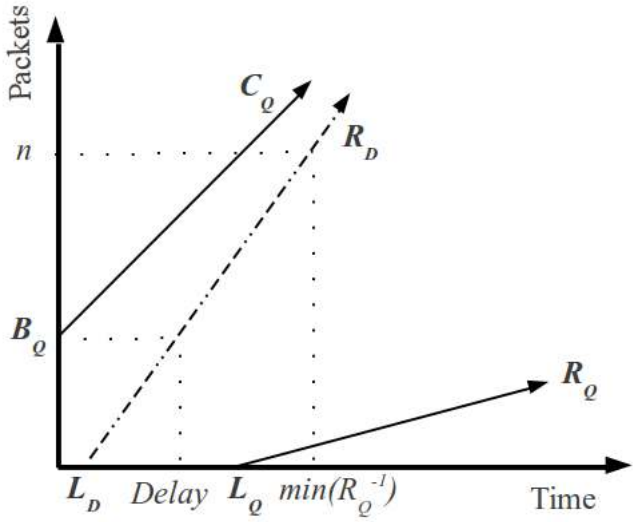



Figure 2.1: QoSoDoS delay analysis. $L_D$ and $R_D$ (dashed) are the DoS/BE network's service latency and rate respectively, which are assured in low delivery probability ($P_D$). $L_Q$ and $R_Q$ are the latency rate assured parameters in high delivery probability ($P_Q$). Packets transmitted by QoSoDoS are bounded by a leaky bucket with burst $B_Q$ and rate $C_Q$. $R_Q = \frac{C_Q}{\alpha \cdot n}$, where $\alpha \geq 1$ is a transmission rate relaxation parameter, and $n \gg 1$ is the number of required packet retransmissions to assure packet delivery in probability $P_Q$.

QoSoDoS is designed to implement a (high reliability) QoS network service over a (low reliability) DoS/BE network service. We model both QoS and DoS/BE network services as *latency-rate (LR) servers* [18]. During a DoS attack, we assume that the best-effort network maximal latency is $L_D$ and minimal rate is $R_D$. For QoS, we require QoSoDoS to ensure a bounded latency $L_Q \geq L_D$ and rate $R_Q \leq R_D$. In addition, in DoS/BE we assume (low) delivery probability $P_D$ ($0 \lesssim P_D \ll 1$), i.e. high probability for packet loss, and in QoS we assume (high) delivery probability $P_Q$ ($0 \ll P_Q \lesssim 1$). The rate we can assure cannot exceed transmitting $n$ packets at the maximum rate, $R_D$, i.e. $R_Q \leq \frac{R_D}{n}$. However, limiting the transmission rate of QoSoDoS to $C_Q$, then for some relaxation parameter $\alpha \geq 1$ it holds that:

$$R_Q = \frac{C_Q}{\alpha \times n},\ \alpha \geq 1 \tag{2.2}$$

To bound the QoSoDoS negative effect on the network, QoSoDoS transmissions are policed. Specifically, QoSoDoS implements a *Leaky Bucket traffic shaper* [18]. The Leaky Bucket model describes a bucket, with a capacity for holding up to $B$ units and a leaking hole which leaks units at a rate of $C$ units every time unit. Flow into the bucket, in rate higher than $C$, fills the bucket up to its capacity - $B$. Any additional packets are poured out, i.e. discarded. In the network QoS analogy, $C_Q$ represents the rate by which packets are transmitted by QoSoDoS over the network, and $B_Q$ represents the burstiness, i.e. the number of packets that can be queued in the LR server. The $B_Q$ parameter affects two things: the buffer required for packet transmission and the delay for the *entire* burst delivery. Specifically, the delay for leaky bucket traffic is bounded by:

$$Delay \leq L_D + \frac{B_Q}{R_D} \tag{2.3}$$

In Table 2.1 we compare the DoS/BE and QoS parameters.

| **Parameter** | **DoS/BE** | **QoSoDoS** |
|---|---|---|
| Latency | $L_D$ | $L_Q \geq L_D$ |
| Rate | $R_D$ | $R_Q = \frac{C_Q}{\alpha \times n},\ \alpha \geq 1$ |
| Packet delivery probability | $0 \lesssim P_D \ll 1$ | $P_D \ll P_Q \lesssim 1$ |

Table 2.1: Comparison between best-effort and QoSoDoS latency-rate server and packet loss parameters (see figure 2.1). $n = \lceil \log_{(1-P_D)}(1-P_Q) \rceil$ is the number of retransmissions required to assure successful delivery in high probability $P_Q$ while packets transmission is made with low delivery probability $P_D$.

### 2.2.2 The Delivery Probability

We now justify the assumption of bounded delivery probability, by presenting a reasonably simplified analysis of a simple queue. To that end we first analyze bounds for the delivery probability $P_D$ using a birth-death process – $M/M/1/K$ queue model – as depicted in Figure 2.2, i.e. a single size limited drop-tail queue serving multiple clients. If some

packet arrives when the queue is full, it is dropped. We use $R$ to denote the arrival rate, and $r$ for the service rate. $R$ consists of both the attacker's and the legitimate clients' rate. We assume that $R \gg r$.

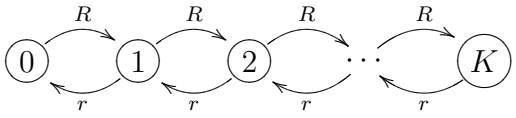



Figure 2.2: Birth-death process ($M/M/1/K$ queue) with $R$ and $r$ as the arrival and service rates.

$\pi_i$ is the probability that the queue contains $i \leq K$ packets. According to PASTA principle, the loss probability is $\pi_K$, hence the loss rate is $R \cdot \pi_K$. In general $R \cdot \pi_i = r \cdot \pi_{i+1}$, hence

$$\pi_i = \left(\frac{R}{r}\right)^i \cdot \pi_0$$

Since $\sum_{i=0}^{K} \pi_i = 1$, then:

$$\pi_0 = \frac{\frac{R}{r} - 1}{\left(\frac{R}{r}\right)^{K+1} - 1}$$

Consequently, the loss probability is:

$$\pi_K = \left(\frac{R}{r}\right)^K \cdot \frac{\frac{R}{r} - 1}{\left(\frac{R}{r}\right)^{K+1} - 1} \to 1 - \frac{r}{R}$$

As we are interested in the delivery probability rather than the loss probability, i.e. $1 - \pi_K$, we get that:

$$P_D \to \frac{r}{R} \tag{2.4}$$

Note that in the analysis if $K$ goes to infinity then $P_D \to \frac{r}{R}$; in practice, the numbers converge very quickly. For $\frac{R}{r} \geq 10$ the equation is correct for $K = 1$ within $\epsilon < 0.001$. For smaller ratios, e.g. $\frac{R}{r} = 1.1$, larger queue sizes of $K \leq 46$ are required, however, the minimal delivery for $K = 1$ is high by itself and larger than 81%. The higher the $\frac{R}{r}$ ratio, the lower the minimal delivery probability, however, the required queue size decreases as well.

In Figure 1.1 we present experimental results supporting the above analysis, and com-

paring the theoretical rate with the rate measured empirically. The bottleneck routers are the default routers in the DETERlab testbed [16], that is FreeBSD machines, employing FIFO scheduling, and 500 slots queue size.

To further emphasize our argument, we will use a simple example. Assuming a bottleneck link of 1Gbps, to produce a BW-DDoS attack in which packet delivery probability drops to 1/100, the attacker should produce roughly 100 times the link capacity, i.e. $100 \times 1Gpbs = 100Gbps$ (the size of the largest bandwidth attack reported to date [12]). Assuming a single strong bot produces 1Mbps of attack traffic, the attacker requires at least 100,000 strong bots. The above scenario also assumes that all attack bandwidth clogs the same bottleneck link, and is not routed via different paths.

We further argue that even under harder conditions, where the probability drops below $P_D$, our model can still assure QoS, only that we implicitly achieve a lower QoS delivery probability ($P_Q$). For example, assuming $P_D = 0.01$, $P_Q = 0.99$ and $P_E = 0.005$, we effectively get packet delivery assurance of 90%, instead of the required 99%, which is still very high.

### 2.2.3 Confronting Network Outage

Network failures which completely prevent communication might also happen, as well as BW-DDoS attacks on a much larger scale than initially estimated using $P_D$. In either case, we would like to refrain from the futile congestion of the network. We therefore identify two cases. The first case is a network failure where the effective packet delivery probability is $P_E = 0$. The second case is when $P_E \ll P_D$, thereby QoSoDoS practically achieves no QoS assurance.

To address these cases we define a state model as described in Figure 2.3. The model has three states: **W**orking, **D**oS and **F**aulty, denoted W, D, and F respectively. The Working state correlates to network low-congestion, in which the vast majority of packets are successfully delivered. The DoS state correlates to network high-congestion, in which delivery probability can drop down to about $P_D$, hence $1 - P_D$ of the packets are

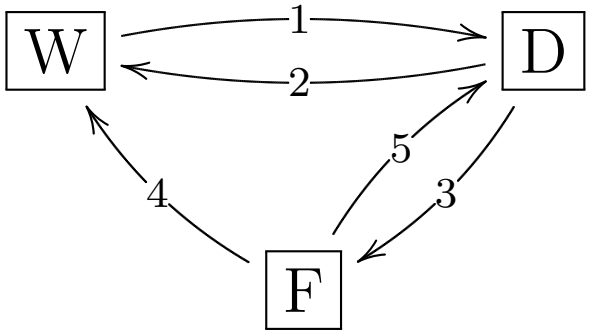



Figure 2.3: QoSoDoS protocol state model and transitions. There are three states: **W**orking, **D**oS and **F**aulty. During the Working state, QoSoDoS uses TCP. During the DoS state, QoSoDoS uses an ART scheme (see Section 2.3). During the Faulty state (where delivery probability $P_E \ll P_D$, e.g. $P_E = 0$), we sample the link at very low rate and occasionally try to resume either the Working or DoS states. Section 2.3.7 contains a design and event description in Table 2.2.

discarded. The Faulty state correlates to network failure or a state in which significantly less than $P_D$ packets are delivered.

While in Working state, QoSoDoS should have a very small effect on the network; during the DoS state, QoSoDoS should employ a retransmission scheme, and during the Faulty state, QoSoDoS should merely sample the network at low rate and see whether the network is back to the Working or DoS states. We describe a design for this mechanism in Section 2.3.7 and empirical results in Section 2.5.5.

## 2.3 QoSoDoS Design

### 2.3.1 Basic Design: Use of ART and TCP

Congestion and BW-DDoS attacks are relatively rare events; most of the time, the network delivers packets with high probability. Therefore, in the design of a new transport mechanism, it is critical to ensure good performance in this typical, benign scenario. One desirable goal would be to obtain comparable performance to that of TCP [64], which is the most established, reliable transport protocol.

To achieve this goal while keeping QoSoDoS as simple as possible, we simply use TCP

to send all packets whenever possible, i.e. initially, and whenever QoSoDoS determines that packet loss probability is sufficiently low. However, QoSoDoS sets a timer for TCP sending; if timeout occurs, and it appears that TCP cannot transmit data, then QoSoDoS suspends TCP usage and begins using a novel *ART (Automated Redundant Transmission)* mechanism, over a connectionless protocol such as UDP.

The ART mechanism is an important component of QoSoDoS, automatically retransmitting each packet, up to $n$ times (as per Eq. 2.1), until the packet is delivered to the destination. ART mechanism should be contrasted with the well-known class of ARQ (Automated Repeat reQuest) algorithms, which are designed for more benign environments with lower loss probability; hence, ARQ algorithms send each packet only once until detecting delivery or loss, while ART algorithms send multiple copies of a packet before receiving indication of loss or delivery.

ART can use different algorithms to schedule retransmissions of a packet. Figure 2.4 depicts a "linear" ART algorithm, which begins by sending a small number of copies, and gradually increases the rate of redundant transmissions. In Section 2.4, we describe several additional ART algorithms.

In addition, we require a mechanism to identify when the DoS attack is over, and resume TCP. The next section describes how QoSoDoS is facilitated with the capability to decrease ART transmission rate and resume TCP. To that end we define an additional parameter, $R_E$, which denotes the *effective* rate by which a fair protocol with congestion control, e.g. TCP [7] or DCCP [46], would transmit packets, *when not under DoS attack*. $R_E$ is set by congestion control protocols as a function of the measured $P_E$ by detecting packet loss. Ideally before the attack $P_E \lesssim 1$. If $P_E$ decreases, a TCP-friendly protocol is assumed to decrease its $R_E$ respectively, thereby increasing $P_E$ back into a stable state of $P_E \lesssim 1$.

On the other hand, during high congestion periods, and especially during DoS attacks, TCP cannot provide any assurance for packet delivery. TCP's behavior of reducing the rate to practically zero does not achieve the desired effect. In such cases, QoSoDoS have

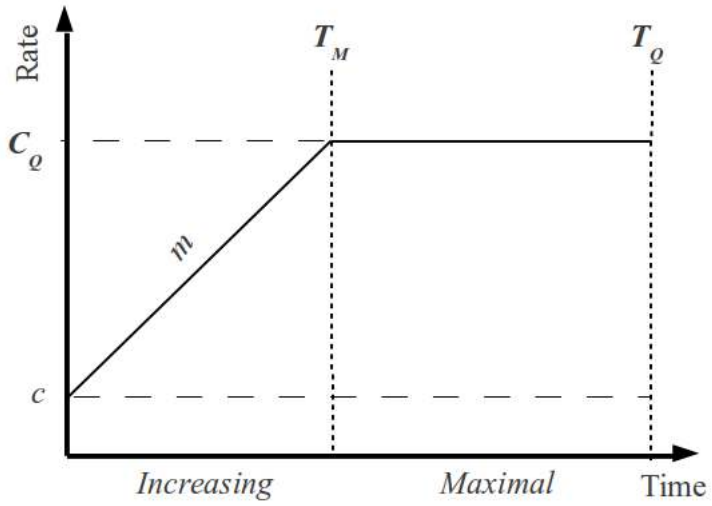



Figure 2.4: "Linear" Automated Redundant Transmission (ART) algorithm, in which the rate increases linearly up to the maximal rate – $C_Q$. $T_Q \geq {R_Q}^{-1}$ is the retransmission deadline, by which all $n$ retransmissions must be made (see Section 2.3.2). $T_M$ is the point in time from which the algorithm sends in rate $C_Q$. $c$ is the initial transmission rate. $m$ is the rate increment slope. The rate as a function of time is $r(t) = \min(m \cdot t + c,\ C_Q)$

a distinct advantage over TCP as it assures communication to the congested destination host.

### 2.3.2 Lowering QoSoDoS' Transmission Rate By Using TCP

A fundamental requirement of QoSoDoS is that it must refrain from self-created DoS, i.e. QoSoDoS clients must refrain from aimlessly congesting the network, especially when no attack is present. We therefore facilitate QoSoDoS with such mechanisms, preventing it from self-initiating DoS attacks, as well as recovering from the high rate transmission of ART. In this section we describe both the usage of TCP within the QoSoDoS core design, as well as relaxing the use of ART's high rate transmissions back to TCP.

As stated in section 2.3.1, QoSoDoS prefers transmitting messages using TCP rather than ART. To achieve this, and still maintain the assured QoS, QoSoDoS uses a TCP user-timeout, which we refer to as $T_T$ ($T$ for time or timeout and subscript $_T$ for TCP). $T_T$ serves as an indicator of the need to suspend TCP and begin transmitting using an ART scheme. $T_T$ limits the time between a TCP packet transmission and its successful delivery.

$T_T$ must be large enough to allow at least the transmission of a single packet, taking into consideration an occasional packet loss and the need for retransmission. Hence, $T_T$ must have an order of *at least* a few RTTs. As we want QoSoDoS to begin with TCP, we refine $L_Q$ (the assured latency) definition to include $T_T$, hence:

$$L_Q = L_D + T_T \tag{2.5}$$

A reasonable value for $T_T$ should be around a few seconds. Larger values allow TCP to perform under some states of congestion. Nevertheless, a too-large value will prolong $L_Q$, as well as the recovery time after an attack as described in Section 2.3.5.

To achieve the design goal of relaxing the high retransmission rate, we designed QoSoDoS as follows; instead of assuring a rate based on transmitting all $n$ packets at the highest rate $C_Q$, thereby minimzing the time for packet acceptance, we relax the time requirement and allow packets to be transmitted over a longer period of time, yielding a lower assured rate of $R_Q = \frac{C_Q}{\alpha \times n},\ \alpha > 1$. Finally, we define $T_Q$, a single packet's assured transmission deadline (i.e. QoS assurance time) as:

$$T_Q \geq {R_Q}^{-1} \tag{2.6}$$

For the first packet and in the worst case, where for all transmissions we have to send all $n$ packets, $T_Q = {R_Q}^{-1}$. For the typical case, where the packet is received prior to sending all $n$ packets, we might have more time for the transmission, hence $T_Q > {R_Q}^{-1}$.

### 2.3.3 Lowering Average Transmission Rate of ART Algorithms

Since $R_E$ is the average rate in which a TCP-friendly flow would transmit, then reducing the average transmission rate down to $R_E$ should cause most packets to be delivered. As we cannot determine $R_E$ a priori, we can split packet transmission during the ART algorithm execution into two time periods, namely *increasing* and *maximal*, similar to

the depiction in Figure 2.4. In the increasing period, ART increases the transmission rate *starting at very low rate* – we consider $0 \leq c \leq RTT^{-1}$ – up to $C_Q$. Next, ART should continue transmitting at the maximal allowed rate – $C_Q$ – until the deadline $T_Q$ is reached.

The average transmission rate produced by ART algorithms can be calculated as $\frac{N}{T}$ where $N$ is the expected number of retransmitted packets and $T$ is the expected transmission time. To simplify the calculations of $N$ (Eq. 2.8) and $T$ (Eq. 2.9), we define the function $art(x)$ to return the transmission time of packet $x$. $art(x)$ can be extracted from:

$$\int_{t=0}^{art(x)} r(t)\,dt = x,\ 1 \leq x \leq n \tag{2.7}$$

where an $r(t)$ is an ART rate function. For example, for the "linear" ART $r(t) = \min(m \cdot t + c, C_Q)$, hence:

$$art(x) = \begin{cases} \frac{-c+\sqrt{c^2+2mx}}{m} & \text{for} \quad x \leq \int_0^{T_M}(mt+c)\,dt \\ \\ \frac{m\cdot x+c-\sqrt{c^2+2\cdot m\cdot T_M}}{(C_Q-T_M)m} & \text{for} \quad x > \int_0^{T_M}(mt+c)\,dt \end{cases}$$

As there is an additional RTT between the time that an accepted packet is acknowledged at the source, we inspect the number of packets added between $art(x) + RTT$ and its previous packet $art(x-1) + RTT$, hence we define:

$$f(x) = \int_{t=art(x-1)+RTT}^{art(x)+RTT} r(t)\,dt,\ art(0) = -RTT$$

The expected number of transmitted packets is therefore:

$$N = \sum_{x=1}^{n}(1-P_D)^{x-1} \cdot f(x) \tag{2.8}$$

where $(1-P_D)^x$ is the probability that neither of the previous packet was accepted by the destination, and $f(x)$ is the additional number of packets the ART will have transmitted

*before* an acknowledgement is accepted.

Equation (2.9) defines the expected transmission time $T$ for each ART transmission, hence:

$$T = \sum_{x=1}^{n} (1 - P_D)^{x-1} \cdot (art(x) - art(x-1)),\ art(0) = 0 \tag{2.9}$$

Figure 2.5 depicts the average rate vs. the delivery probability for a linear ART. Note that the percentage of rate usage is higher for higher rates. This is the result of the amount of packets sent at any RTT, and especially in the initial RTT, which produces the minimal load inflicted by each QoSoDoS client. Inspecting Equation 2.2: $R_Q = \frac{C_Q}{\alpha \cdot n}$, in which the higher $C_Q$, the more packets are sent per RTT, we note that using even higher rates under the same configuration will cause an even higher average rate usage. Hence, we conclude that ART's average rate should be carefully controlled by making sure that $\alpha > 1$ so that $T_Q \gg \frac{n}{C_Q}$, to avoid average transmission rate equal to $C_Q$, as well as keeping $T_Q \gg RTT$ to avoid high minimal transmission rates. Other configurations may force limiting the amount of allowed QoSoDoS clients.

### 2.3.4 Number of Clients vs. Delay Trade-off

Examining Equation (2.4), Section 2.3.3, and Figure 2.5, we note that we must make sure that after an attack, the remaining congestion which is inflicted by QoSoDoS will reduce until QoSoDoS can safely and successfully resume TCP.

To simplify our discussion we regard a model as depicted in Figure 2.6, assuming that all legitimate clients are similar. We therefore want to bound the successful delivery probability as follows. The worst case is when all clients transmit packets at the maximal rate $C_Q$, that hence have the same probability for being queued by the network's routers, and therefore have the same probability for being forwarded to the destination host. If $m$ clients are transmitting at $C_Q$ and their effective rate should have been $R_E$, had they used TCP, then $P_E \geq \frac{m \times R_E}{m \times C_Q} = \frac{R_E}{C_Q}$. The number of packets needed to assure transmission is therefore: $n' = \log_{(1-P_E)}(1-P_Q)$. We assume $P_E \gg P_D$, hence $n \gg n'$. This suggests that

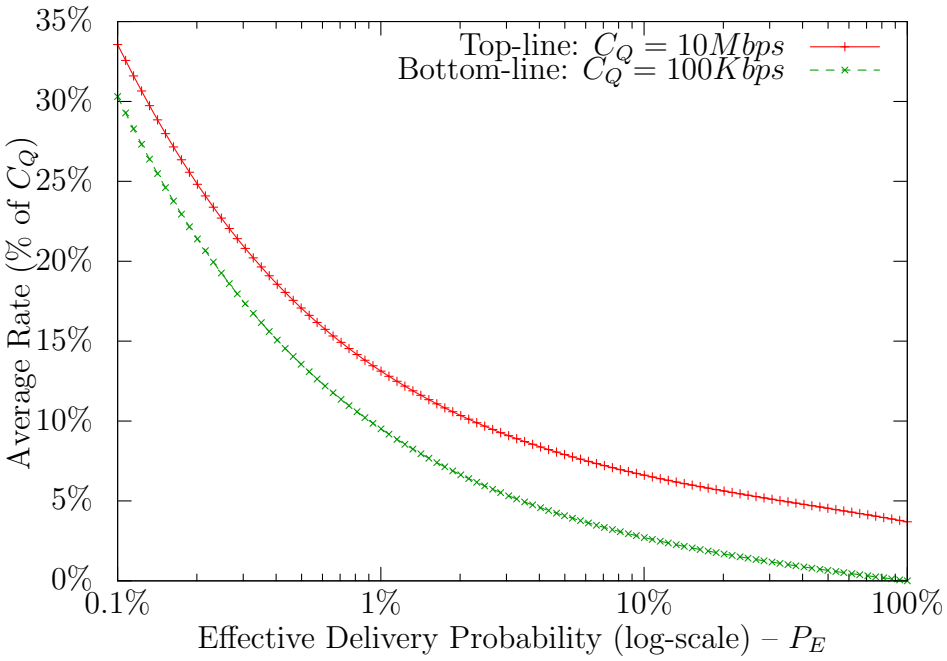



Figure 2.5: Average rate (as percent of $C_Q$) vs. delivery probability (log scale) during a single ART transmission using the "linear" ART algorithm (see Figure 2.4). Both lines are produced using parameters taken from Table 2.5. For the top line $C_Q = 10Mbps$ and for the bottom line $C_Q = 100Kbps$. The average rate when $P_E = P_D$ is $\approx 1/3$ of $C_Q$. Note that in most cases QoSoDoS will try to resume TCP between ART transmissions (see Section 2.3.5), therefore further reducing the average rate. The average rate can be further reduced using a larger delay, i.e. larger $\alpha$ parameter, as described in Section 2.3.4.

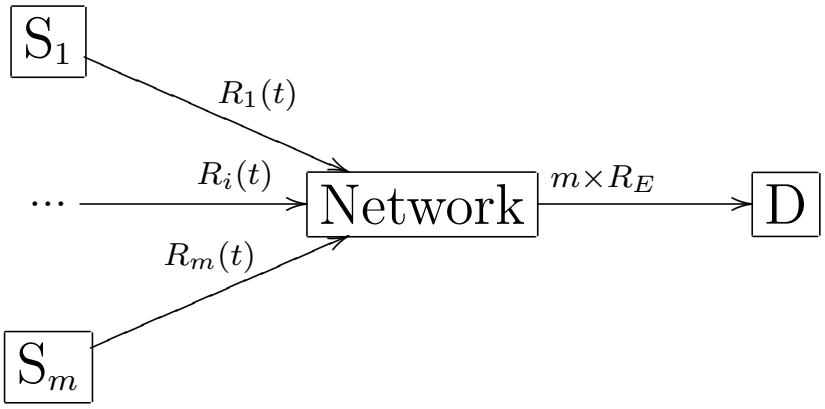



Figure 2.6: a simplified transmission model from multiple senders to a single destination. $S_1..S_m$ represents $m$ senders, $D$ represents the destination, $R_i(t) \leq C_Q,\ 1 \leq i \leq m$ represents the $i'th$ sender's transmission rate at time $t$ and $m \times R_E$ represents the available rate by which the network transmits packets to the destination host.

much fewer packets need to be transmitted in order to assure delivery, and therefore much less time should pass until a packet's acceptance. We would like to have a correlation to the results presented in Figure 2.5, such that the average rate will be reduced when the attack is over, down to a minimum congestion which will allow resumption of TCP. The congestion should therefore be a function of the number of clients. We would expect that as long as the number of QoSoDoS clients is policed, self-DoS can be avoided.

To provide the number of allowed clients, we examine Equation 2.4. The delivery probability should stabilize on $P_E = \frac{R_N}{|C| \cdot R_C(P_E)}$, where $R_N$ is the network rate, $|C|$ is the number of clients and $R_C(P_E)$ is the average rate of a client as a function of effective delivery probability, as depicted in Figure 2.5. Note that $R_C(P_E)$ decreases as $P_E$ increases, i.e. $P_E \propto R_C(P_E)^{-1}$. As $P_E$ should gradually increase, we would like to find out the parameters' values in which $P_E$ stabilizes to a probability which is high enough to allow successful resuming of TCP, hence we argue that $P_E$ should stabilize on a value larger than 0.7, which produces approximately 50% probability for two consecutive transmissions to succeed. Such two transmissions correlate to a SYN (answered with a SYN/ACK) followed by an ACK containing data. In the above equation the $R_N$ is given, hence we can control the stable $P_E$ either by reducing the number of clients, or reducing $R_C(P_E)$, which means reducing $R_Q$. Recall that $R_Q = \frac{C_Q}{\alpha \cdot n}$, and since both $C_Q$ and $n$

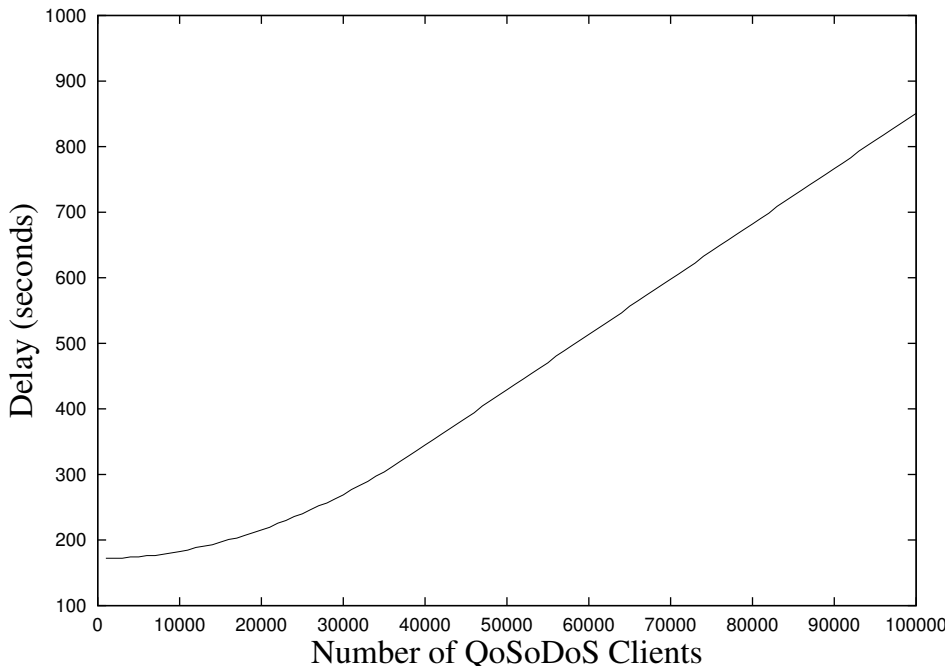



Figure 2.7: Delay vs. Number of Clients. After an attack, the effective delivery probability should stabilize on $P_E = \frac{R_N}{|C| \cdot R_C(P_E)}$. In the above figure we require that $P_E$ stabilize on $P_E \geq 0.7$. Note that the more clients are supported, the worse the assured delay. Specifically, the figure demonstrates that by policing the number of QoSoDoS clients and their configuration, self-inflicted DoS can be avoided.

(Equation 2.1) are given, ultimately we can only increase $\alpha$, thereby decreasing $R_Q$ and consequently $R_C(P_E)$.

Figure 2.7 depicts the relation between the assured delay for a single packet transmission as a function of the number of QoSoDoS clients. The figure shows that by policing the number of clients we can assure resuming of TCP, and more specifically, the more clients QoSoDoS has to support, the worse is the assured delay.

Based on the above results, Figure 2.8 depicts a theoretical macro-view of multiple QoSoDoS clients in a DoS attack scenario where TCP transmissions are aborted by QoSoDoS, followed by ART transmissions and a recovery period until TCP is resumed. Figure 2.9 depicts a correlating theoretical micro-view of a single QoSoDoS client's multiple packet transmissions within the context of Figure 2.8. In our experiments (see Section 2.5), QoSoDoS clients resumed TCP almost immediately after the attack stopped.

Concluding Sections 2.3.3 and 2.3.4, we find that there are three reasons for the actual

rate decrease and the resumption of TCP. The first reason is that the DoS attack is over. The second reason is the ART behavior of decreasing average transmission rate. The third reason is the delivery probability stabilization to a high enough probability. When the attack is over, it implies an increase in legitimate packet delivery probability. Next, the retransmission behavior which lowers the average transmission rate comes into action, since the remaining congestion is created due to ART by legitimate clients. This means that packets reaching the destination host are legitimate. Hence, even in the worst case, where all legitimate clients are transmitting packets at rate $C_Q$, one legitimate packet should be acknowledged by the destination host, causing its sender to reduce its rate. Probabilistically, the higher the transmission rate the higher the delivery probability is, which is supposed to help clients transmitting at higher rates reduce their transmission rate within an RTT. Finally, we make sure that when no attack is present, the network's effective delivery probability will stabilize on a delivery probability which is high enough to enable TCP to perform well enough.

### 2.3.5 Resuming TCP

Before resuming TCP, QoSoDoS must assure that it has enough time for both the re-suspension of TCP, in case it fails or timeouts, and re-initiation of ART, without harming the assured QoS. In order to do so, QoSoDoS inspects the assured parameters, and the actual packet acceptance state. It compares the current time and the next packet's guaranteed time of delivery. This is done by inspecting the next packet in the burst, if such exists, or the minimal time needed for transmission of a potential newly arriving packet. If the time until the guaranteed time of delivery for the next packet is greater than the time needed for re-suspending TCP ($T_T$) and its transmission using ART ($T_Q$), then QoSoDoS can safely try to resume TCP, otherwise, it must continue transmitting in the ART scheme, so that the assured QoS will not be jeopardized. If there are no packets waiting for transmission, i.e. no burst, we can resume TCP immediately, as $L_Q = L_D + T_T$, and enough time exists between the next packet arrival and the QoS

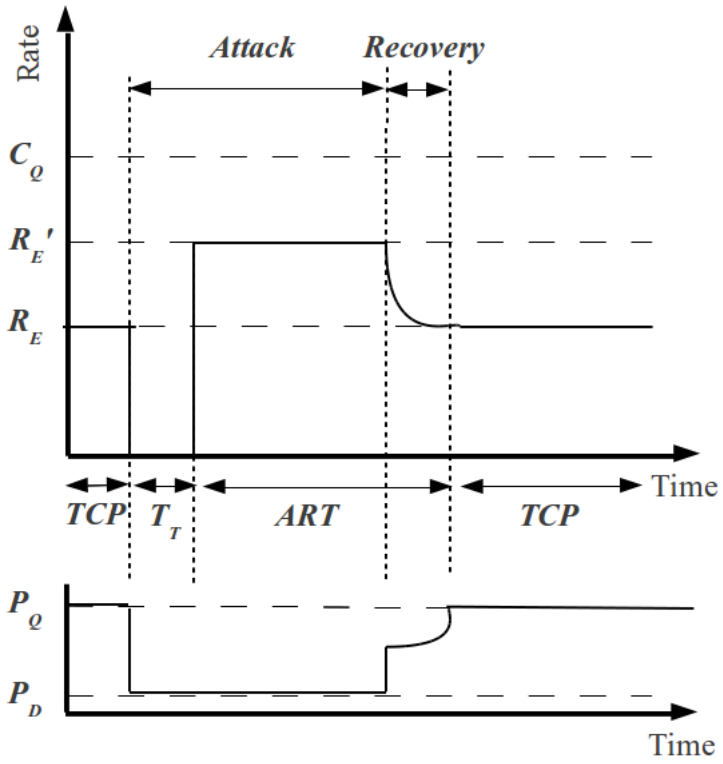



Figure 2.8: QoSoDoS average rate vs. time (theoretical macro-view based on Figures 2.5 and 2.7). Top figure represents rate vs. time of multiple QoSoDoS clients, and the bottom figure is a correlating single packet's effective delivery probability ($P_E$). In the top figure, the solid line represents the average rate by which legitimate clients would transmit packets. $R_E$ is the effective average rate by which TCP would transmit packets under congestion control (w/o attack). At the beginning of the DoS attack TCP practically stops transmitting data. $T_T$ is QoSoDoS' TCP-timeout interval, which, when it expires, results in the abortion of TCP and initiation of ART transmission. When the DoS attack is over, QoSoDoS starts a recovery period. At time $t$ QoSoDoS resumes TCP.

commitment.

Finally, merely *trying* to resume TCP is helpful in reducing the average QoSoDoS rate cf. to ART rate (see Figure 2.5), since it adds an interval $T_T$, in which very few packets are sent.

### 2.3.6 QoSoDoS: The Big Picture

QoSoDoS basically works in two modes derived from the network state, namely low and high congestion modes. In low congestion, QoSoDoS uses TCP as is, thereby taking advantage of TCP mechanisms such as congestion and flow control. In high congestion,

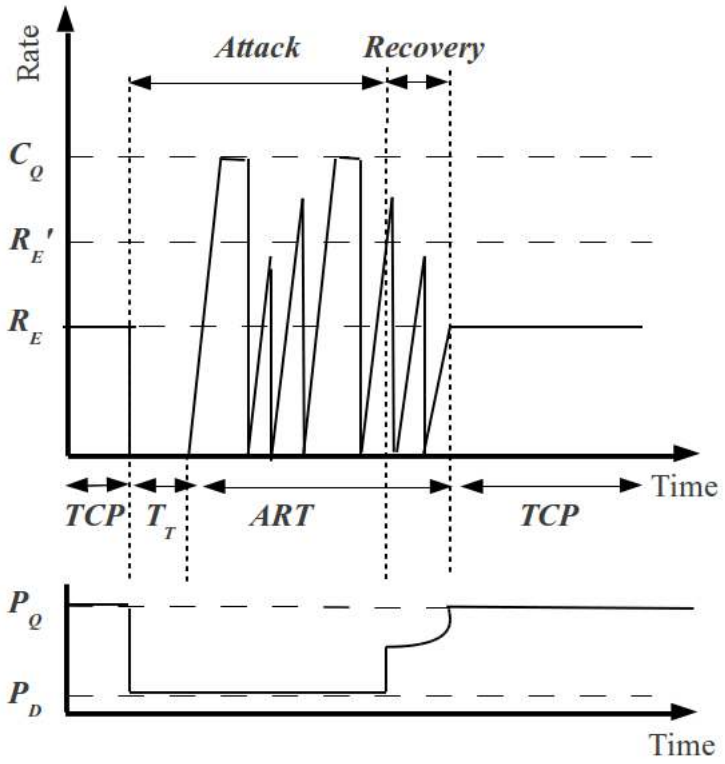



Figure 2.9: QoSoDoS' multiple packet transmission rate over time (theoretical micro-view based on Figures 2.5 and 2.7). The top figure represents rate vs. time of a single QoSoDoS client, and the bottom figure represents a correlating single packet's effective delivery probability ($P_E$). In the top figure, the solid line is the rate by which legitimate clients transmit packets. At $T_T$, QoSoDoS aborts TCP transmission and initiates ART. $R_E$ is the rate by which TCP would have transmitted (w/o attack). Each distinct climb is a different packet transmission (see figure 2.4). At time $t$ QoSoDoS resumes TCP. In the experiments described in Section 2.5 most transmissions returned instantaneously to TCP.

in which packet delivery probability is low and TCP cannot function properly, QoSoDoS uses ART.

The principle on which QoSoDoS relies during high congestion periods is that packet delivery is essentially statistical, even under BW-DDoS attacks. During an attack, the attacker tries to clog the victim host's bandwidth by transmitting numerous futile packets and thus preventing legitimate packets from reaching that host; therefore most (legitimate) packets sent to the victim host are dropped. Nevertheless, probabilistically, once in a while a legitimate packet should come through and reach its destination. During an attack, the probability of packet successful delivery (assuming independence, and same

packet size) drops down to about $P_D \sim \frac{R_N}{R_A+R_C}$, where $R_N,\ R_C,\ R_A$ are network, (legitimate) client and attacker rates respectively. See analysis in Section 2.2.2.

The following is a numerical example. Assume $R_N = R_C \cdot 100$ and $R_A = R_C \cdot 999,999$, hence $P_D = 0.0001$. Assume $C$ wants its packet delivered with $P_Q = 99\%$ certainty, hence $C$ needs to retransmit the packet $n = \lceil \log_{(1-P_D)}(1-P_Q) \rceil = 46,050$ times. If $C$ can transmit $C_Q = 1,000$ packets/second, $C$ can assure delivery within $T_Q = 46.05$ seconds. $C$ will optimistically start transmission using TCP, in hopes that it will succeed in delivering data. After some time, say $T_T = 3$ seconds, $C$ aborts TCP, and starts using ART. To relax the average ART rate, for support of multiple QoSoDoS clients, $C$ sets $\alpha = 1.5$, hence all $n$ retransmissions end by $T_Q = \frac{46,050 \times 1.5}{1,000} = 70$ seconds. Assuming independence, $C$'s packet should get accepted after an expected $\frac{1}{P_D} = 10,000$ retransmissions, implying shorter latency than 70 seconds, and a higher expected assured rate.

### 2.3.7 Network Outage Handling Design

As described in Section 2.2.3, QoSoDoS deals with network outages by trying to estimate whether the current network state is Working, DoS or Faulty (see Figure 2.3). The above sections describe the transition between the Working and DoS States, in which TCP and ART are used respectively, and what triggers the transition. The Faulty state is used when ART fails to deliver a few consecutive packets. In such a case there is a large probability that either $P_E = 0$, i.e., the network actually cannot delivery any packets, or that $P_E \ll P_D$, which by design we consider a network outage. In the Faulty state we want to stop unnecessary retransmissions and merely sample the network and identify when it becomes available again.

We define two parameters: $\phi_{init}$ and $\phi_{sample}$. $\phi_{init}$ denotes the Faulty state's initialization condition, and constitutes the number of consecutive ART transmission failures. The probability that no packet was received after $\phi_{init}$ ART transmissions is $(1 - P_Q)^{\phi_{init}} = (1 - P_D)^{n \cdot \phi_{init}}$ which suggests that $P_E \ll P_D$. Hence if $\phi_{init}$ consecutive ART transitions failed, QoSoDoS enters the Faulty state; see edge 3 in Figure 2.3).

| # | **Description** |
|---|---|
| 1 | $T_T$ (pre-defined TCP timeout) has expired (see Section 2.3.2). |
| 2 | QoSoDoS had a successful transmission and more than $T_T$ time exists before next (assured) transmission must take place. |
| 3 | After $\phi_{init}$ failed ART transmissions, change to Faulty state. See also Event #5. |
| 4 | TCP connection reestablished (resume Working state). |
| 5 | After $\phi_{sample}$ attempts to resume TCP (Event #4), try sending a *single* packet using QoSoDoS. If that single packet delivery failed, resume Faulty state (Event #3). |

Table 2.2: QoSoDoS events description per state model depicted in Figure 2.3.

$\phi_{sample}$ denotes the Faulty state's sampling rate. When in Faulty state, QoSoDoS tries to estimate when the network failure is over and packet delivery can resume. Since we specifically assume failure and not DoS, we sample the link by trying to resume TCP. We retry to establish TCP connection $\phi_{sample}$ times, and each attempt is $T_T$ in duration (see Section 2.3.2 for definition). TCP sends just a few SYN packets while trying to establish a new connection, therefore it should introduce only minimal congestion to the network. After $\phi_{sample}$ attempts to establish a TCP connection, QoSoDoS tries to send a single packet in ART (edge 5 in Figure 2.3). If packet delivery is not successful then the Faulty state is reinitialized, without the additional $\phi_{init} - 1$ retries made upon first transition to the Faulty state (edge 3 in Figure 2.3). This mechanism further prevents QoSoDoS from self-creating DoS and prevents unnecessary packet transmissions on network failures. Table 2.2 provides a summary for the transitions. See Section 2.5.5 for experimental results.

### 2.3.8 Algorithm Design

Algorithm 1 describes a transmission algorithm, based on the QoSoDoS model. Algorithm 1 uses a general ART transmission algorithm, generically used by calling the $ART()$ function in line 23 of the algorithm, which is described in Algorithm 2. The actual algorithm, that is the NP and SP implementations used inside the ART algorithm, may vary; in Section 2.4 we discuss a few possible implementations, such as the "linear"

algorithm depicted in Figure 2.4. We also compared these different implementations empirically in Section 2.5.

Algorithm 1 begins by setting the number of consecutive fails to 0 and the initial state to Working, meaning that we optimistically start by using TCP.

In line 3 we dequeue a packet from the *queue* or wait until a packet is available for dequeuing. Once we have dequeued a packet we test whether we are in the DoS state and have enough time to try and resume the Working state, thereby trying to send the packet using TCP. Initially the *state* is Working, hence the $if$ in line 4 fails as soon as we test whether the *state* is DoS. This implies that the values of $i$ and $t_{art}$ are only tested if the *state* is DoS, which is an important fact as both $i$ and $t_{art}$ are only initialized in lines 13-16 when we switch to the DoS state.

Based on line 5, we will try using TCP if either $state = Working$ or $fails > \phi_{init}$. The latter condition means that we got to this line when $state = Faulty$ and we are trying to sample the link and to resume TCP. Lines 6-9 describe the process of trying to send the packet using TCP. In case TCP did not get an ACK within $T_T$ we switch to the DoS state, that is, unless the state is Faulty and we do not yet need to activate Event #3. In case the state is Faulty and it is time to sample the link using the DoS state, then line 14 makes sure that we will enter the DoS state for one transmission only and reenter the Faulty state in case it fails.

Lines 17-30 describe the UDP-based retransmissions that take place during the DoS state. The DoS state retransmissions are timed based on the ART algorithm, defined in Algorithm 2, which returns the amount of time that the algorithm should wait before the $j$'th retransmission has to take place.

## 2.4 ART Algorithms

In this section we present several ART algorithms we have developed which are comparable using various measures, which we tested as reported in Section 2.5.

**Input**:

| | |
|---|---|
| $C_Q$ | // Max allowed transmission rate |
| $R_Q$ | // QoSoDoS assured rate |
| $T_T$ | // QoSoDoS timeout for using TCP |
| $n$ | // $\equiv \lceil \log_{(1-P_D)}(1-P_Q) \rceil$ (max retrans.) |
| $\phi_{init}, \phi_{sample}$ | // Parameters for state changes |
| $queue$ | // Transmission queue shared w/sender |

**Data**:

| | |
|---|---|
| $T_Q$ | // $\equiv R_Q^{-1}$ (retrans. deadline) |
| $state$ | // Working, DoS or Faulty |
| $pkt$ | // Packet to transmit |
| $t_{art}, t_{pkt}$ | // ART/packet transmission start time |
| $i$, $j$ | // #trans. and #retrans. counters |
| $fails$ | // Consecutive failures counter |

```
1  fails ←0, state ←Working;
2  while True do
3      pkt ← Dequeue(queue);
4      if state=DoS and (now() − t_art) + T_T + T_Q < i · T_Q then state ←Working;
5      if state=Working or fails > φ_init then
6          t_pkt ← now();
7          Async-TCP-Transmit(pkt);
8          Wait for TCP abort or pkt ACK or (now() − t_pkt) = T_T;
9          if pkt ACKed then fails ←0;
10         if pkt not ACKed then
11             Abort TCP connection;
12             if satate=Working or (fails mod φ_sample) = 0 then
13                 i ← 1;
14                 if state ≠ Working then i ← φ_init − 1;
15                 state ←DoS;
16                 t_art ← now();
17     if state =DoS then
18         j ← 0;
19         t_pkt ← now();
20         while pkt not ACKed and j < n and (now() − t_pkt) < T_Q do
21             UDP-Transmit(pkt);
22             j ← j + 1;
23             sleep(ART(now() − t_pkt, j, n, T_Q, C_Q));
24         end
25         if pkt not-ACKed then
26             Notify: Failed to deliver packet;
27             fails++;
28             if fails ≥ φ_init then state=Faulty;
29         if pkt ACKed then fails ←0;
30         i ← i + 1;
31 end
```

**Algorithm 1**: QoSoDoS' packet scheduling algorithm. $ART()$ is an ART algorithm (see Section 2.4).

| Type | Param | Description | Section |
|---|---|---|---|
| **Prob** | $P_D$ | DoS-prone delivery prob (low) | 2.2, 2.2.2 |
| | $P_Q$ | QoSoDoS delivery prob. (high) | 2.2 |
| | $P_E$ | Effective delivery prob. $P_D \leq P_E$ | 2.2 |
| | $P_{N/A}$ | $\min(1, \frac{R_N}{R_A})$ (attacker induced delivery prob.) | 2.5.1 |
| **Rate** | $R_D$ | DoS-prone transmission rate (high) | 2.2.1 |
| | $R_Q$ | QoSoDoS transmission rate (low) | 2.2.1 |
| | $R_E$ | Effective trans. rate | 2.3.1 |
| | $R_N$ | Network rate (bottleneck's bandwidth) | 2.3.6 |
| | $R_A$ | Attacker rate | 2.3.6 |
| | $C_Q$ | Maximal QoSoDoS(ART) transmission rate | 2.2.1 |
| **Latency** | $L_D$ | DoS-prone network latency (low) | 2.2.1 |
| | $L_Q$ | QoSoDoS network latency (high) | 2.2.1 |
| **Time** | $T_T$ | QoSoDoS timeout for using TCP | 2.3.2 |
| | $T_Q$ | Single packet trans. time (Eq. 2.6) | 2.3.2 |
| | $T_M$ | The time when $r(t)$ reaches $C_Q$ | 2.3.1,2.3.3 |
| **Misc.** | $n$ | Required no. of retransmissions (Eq. 2.1) | 2.2 |
| | $\alpha$ | Relaxation/delay parameter (Eq. 2.2) | 2.2.1 |
| | $\phi_{init}$ | Faulty state initialization parameter | 2.3.7 |
| | $\phi_{sample}$ | Faulty state sampling parameter | 2.3.7 |
| **Func.** | $r(t)$ | Rate as a function of time | 2.3.1,2.3.3 |
| | $R_C(P_E)$ | Average rate as a function of $P_E$ | 2.3.4 |
| | $art(x)$ | ART's trans. time of packet $1 \leq x \leq n$ | 2.4(Alg.2) |
| | $NP(x)$ | *Next Packet*'s $(x+1)$ transmission time | 2.4 |
| | $SP(t)$ | *Sum of Packets* (supposedly) sent by time $t$ | 2.4 |

Table 2.3: QoSoDoS Notation Summary Table.

We define Algorithm 2, which for convenience purposes uses two functions, namely $NP$ and $SP$. $NP$ gives the *Next Packet*'s transmission time, relative to the beginning of a specific ART transmission (see Algorithm 1). $NP$ assumption is that it is called in conjunction to sending a packet. $SP$ returns the *Sum of Packets* that were supposed to be transmitted by time $t$. Both $NP$ and $SP$ receive $n$, $T_Q$ and $C_Q$ as input, in order to calculate each algorithm's parameters, as described later.

```
1 ART(t, i, n, T_Q, C_Q)
  Input:
    t        // time since current ART transmission began, 0 ≤ t ≤ T_Q.
    i        // transmitted packet counter.
    n        // maximal packet retransmissions.
    T_Q      // deadline for n retransmissions.
    C_Q      // maximal permitted transmission rate.
2 if i < SP(t, n, T_Q, C_Q) then
      /* Send now                                                    */
3     return 0;
4 end
  /* Send in NP seconds                                             */
5 return NP(t, n, T_Q, C_Q);
```

**Algorithm 2**: ART next transmission algorithm is used by all transmission algorithms. $NP$ returns the *Next Packet* transmission time and $SP$ return the *Sum of Packets* that were supposed to be sent by time $t$. The implementations of $NP$ and $SP$ are algorithm-specific (see Section 2.4).

The algorithms are presented along with rate vs. time diagrams. The area beneath the function is the number of packets sent, i.e. the area in the time interval $0 \leq t \leq T_Q$ must be equal to $n$.

### 2.4.1 Flat Algorithm

The "Flat" algorithm is straightforward and is based on retransmitting packets at a constant rate, $\frac{n}{T_Q}$ (see Figure 2.10). Eq. (2.10) and (2.11) define $NP$ and $SP$ for the "Flat" algorithm respectively.

$$NP(t, n, T_Q, C_Q) = \frac{n}{T_Q} \tag{2.10}$$

$$SP(t, n, T_Q, C_Q) = t \cdot \frac{n}{T_Q} \tag{2.11}$$

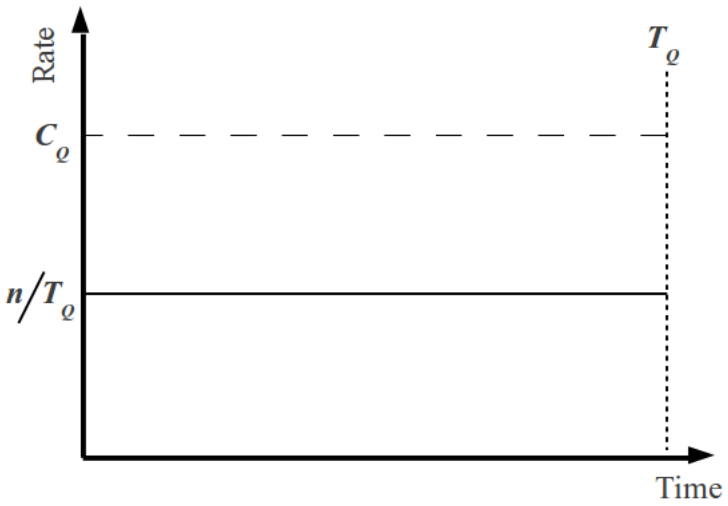



Figure 2.10: "Flat" rate algorithm. $C_Q$ is the maximal allowed transmission rate, while the rate for packet retransmissions is $\frac{n}{T_Q}$. $T_Q$ is the deadline for all $n$ retransmissions.

### 2.4.2 Bulk Algorithms

Bulk algorithms are bi-rate algorithms; that is, they either send at a high or low rate, while making sure that the required sum of messages are sent by the deadline time ($T_Q$).

#### "Bulk At Start" Algorithm

The "bulk at start" algorithm tries to send packets at rate $C_Q$ until a packet is acknowledged or all $n$ packets were transmitted. Figures 2.11 depicts "bulk at start," while Eq. 2.12 and Eq. 2.13 define $NP$ and $SP$ respectively.

$$NP(t, n, T_Q, C_Q) = \min\left(\frac{1}{C_Q}, T_Q - t\right) \tag{2.12}$$

$$SP(t, n, T_Q, C_Q) = \min\left(\frac{t}{C_Q},\ n\right) \tag{2.13}$$

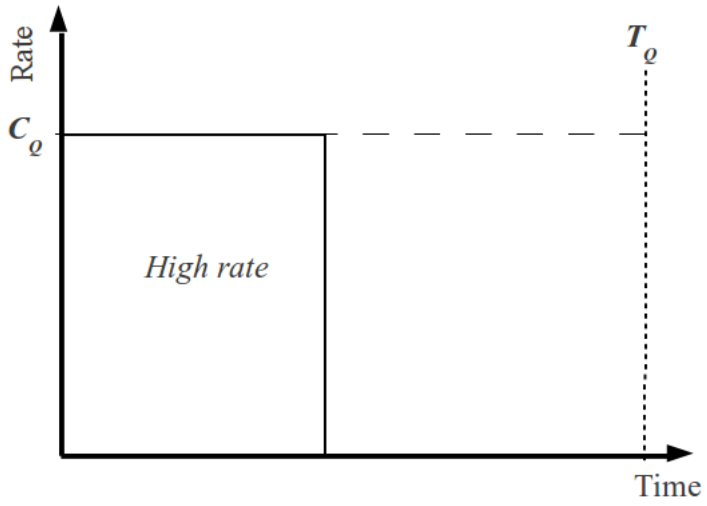



Figure 2.11: "Bulk at start" algorithm starts retransmitting packets at the highest rate $C_Q$, and continues doing so until either the one of the packets is acknowledged or $n$ packets were transmitted. This behavior may send all $n$ packets in less than an RTT, effectively disregarding any potential previously accepted packets (see Section 2.3.2)

### "Bulk At End" Algorithm

"Bulk at end" transmits at a low rate, such as $\frac{1}{RTT}$, at the beginning and switches to the maximal rate $C_Q$ towards the end. Unlike "bulk at start," low rate transmission is used, as it makes little sense to hold off all transmissions until the end, and remain silent for a long period of time. Figure 2.12 depicts "bulk at end."

The "bulk at end" algorithm finds the point in time prior to the deadline ($t \leq T_Q$), in which it must begin transmitting at the maximal rate $C_Q$, using a function $T_M$. At this point the algorithm switches from the low rate $c$, e.g. $RTT^{-1}$, to the maximal rate $C_Q$. The sum of the rectangles' area, i.e. the area of the rectangle at the low rate and the area of the rectangle at the high rate, is equal to $n$. Eq. (2.14), (2.15) and (2.16) defines $T_M$, $NP$ and $SP$ for "bulk at end" respectively.

$$T_M \equiv \frac{n - C_Q \cdot T_Q}{c - C_Q} \tag{2.14}$$

$$NP(t, n, T_Q, C_Q) = \begin{cases} \frac{1}{c} & \text{for} \quad t < T_M \\ \min\left(\frac{1}{C_Q}, T_Q - t\right) & \text{for} \quad T_M \leq t \leq T_Q \end{cases} \tag{2.15}$$

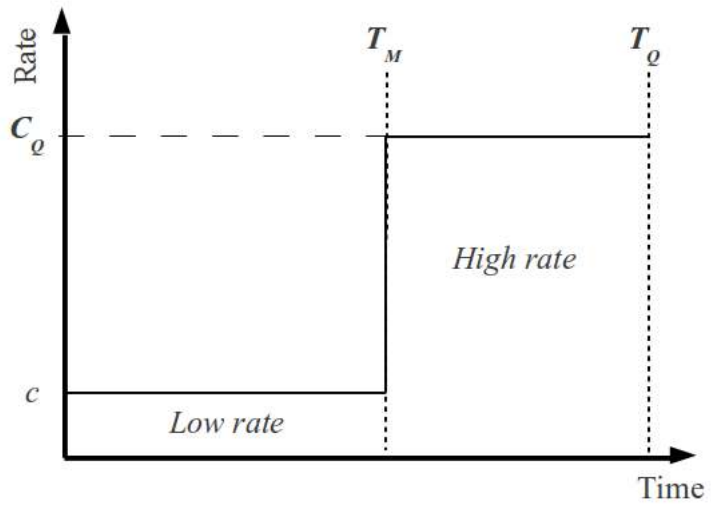



Figure 2.12: "Bulk at the end" algorithm retransmits packets at a low rate $c$, such as $c = \frac{1}{RTT}$, until the time left until $T_Q$ requires it to transmit the remaining packets at $C_Q$, in which it increases its transmission rate respectively.

$$SP(t, n, T_Q, C_Q) = \begin{cases} t \cdot c & \text{for} \quad t < T_M \\ T_M \cdot c + (t - T_M) \cdot C_Q & \text{for} \quad T_M \leq t < T_Q \\ n & \text{for} \quad t \geq T_Q \end{cases} \tag{2.16}$$

### 2.4.3 Linearly Increasing Rate Algorithm

The Linearly Increasing Rate algorithm increases the rate linearly starting at the minimal rate $c$, e.g. $RTT^{-1}$ up to the point where it reaches the maximal rate $C_Q$. Figure 2.4 depicts the Linearly Increasing Rate algorithm, and we use it for parameters reference. As before, our aim is to define the $NP$ and $SP$ functions. To this end we define the rate function, as:

$$r(t) = min\left(C_Q,\ m \cdot t + c\right)$$

We find $T_M$ by using area calculations which denote the number of sent messages. The area under the slope $m$, and the rectangle between $T_M$ and $T_Q$ equals $n$, i.e. $c \cdot T_M +$

$\frac{T_M(C_Q-c)}{2} + (T_Q - T_M) \cdot C_Q = n$. Hence:

$$T_M \equiv \frac{2(n - T_Q \cdot C_Q)}{c - C_Q} \tag{2.17}$$

Based on $T_M$, the slope $m$ is:

$$m = \frac{C_Q - c}{T_M} \tag{2.18}$$

Prior to defining $NP$ we define $NPS(t)$, which denotes the *Next Packet on Slope*, i.e. the next transmission time when $t < T_M$. $NPS(t)$ is the point where the area between $t$ and $NPS(t)$ equals 1 (packet). Based on $\int_t^{NPS(t)}(mx + c)dx = 1$,

$$NPS(t) = \frac{-c + \sqrt{c^2 + m(mt^2 + 2ct + 2)}}{m} \tag{2.19}$$

As the next packet may need to be transmitted *after* $T_M$, i.e. $NPS(t) > T_M$, and since the rate cannot be increased above $C_Q$, we need to flatten the transmission exceeding $C_Q$. We define the function $PR(\tau)$, which denotes the *Packet Remainder*, for any $\tau > T_M$ as $PR(\tau) = \int_{T_M}^{NPS(\tau)}(mx + c)dx$. Hence:

$$PR(\tau) = \frac{m}{2}(\tau^2 - {T_M}^2) + c(\tau - T_M) \tag{2.20}$$

Finally, we construct $NP$ and $SP$ as follows:

$$NP(t, n, T_Q, C_Q) = \begin{cases} NPS(t) & \text{for} \quad NPS(t) \le T_M \\ T_M + \frac{PR(NPS(t))}{C_Q} - t & \text{for} \quad t < T_M < NPS(t) \\ \frac{1}{C_Q} & \text{for} \quad T_M \le t < T_Q \end{cases} \tag{2.21}$$

$$SP = \begin{cases} t\left(\frac{m}{2}t + c\right) & \text{for} \quad t \le T_M \\ T_M\left(\frac{m}{2}T_M + c\right) + \frac{t - T_M}{C_Q} & \text{for} \quad T_M < t < T_Q \\ n & \text{for} \quad t \ge T_Q \end{cases} \tag{2.22}$$

### 2.4.4 Exponentially Increasing Rate Algorithm

The Exponentially Increasing Rate algorithm is similar to the linear algorithm, only that it increases the rate exponentially instead of linearly. We use Figure 2.13 for parameters reference. We define the rate function as follows:

$$r(t) = min\left(c \cdot \gamma^t, C_Q\right)$$

Our initial objective is to calculate the value of $\gamma$, hence we first define $T_M$ in Eq. 2.23.

$$T_M = \log_\gamma(\frac{C_Q}{c}) \tag{2.23}$$

Based on $C\int_0^{T_M}(\gamma^t)dt + (T_Q - T_M)C_Q = n$, we get Eq. 2.24 as follows:

$$\gamma = e^{\left(\frac{C_Q\left(1-\ln\left(\frac{C_Q}{C}\right)\right)-C}{n-T_Q \cdot C_Q}\right)} \tag{2.24}$$

Similar to the Linearly Increasing Rate algorithm, we define $NPE(t)$, the *Next Packet on Exponent* function, and $PR(\tau)$ for $\tau > T_M$, the *Packet Remainder*. Based on $C\int_t^{NPE(t)}(\gamma^t)dt = 1$, we get:

$$NPE(t) = \log_\gamma\left(\frac{\ln\gamma + c\gamma^t}{C}\right) \tag{2.25}$$

$$PR(\tau) = \frac{c\gamma^\tau - c\gamma^{T_M}}{\ln\gamma} \tag{2.26}$$

Finally, we define $NP$ and $SP$ for the exponential algorithm as follows:

$$NP = \begin{cases} NPE(t) & \text{for} \quad NPE(t) \leq T_M \\ T_M + \frac{PR(NPE(t))}{C_Q} - t & \text{for} \quad t < T_M < NPE(t) \\ \frac{1}{C_Q} & \text{for} \quad T_M \leq t < T_Q \end{cases} \tag{2.27}$$

$$SP = \begin{cases} \frac{c\gamma^t - c}{\ln \gamma} & \text{for} \quad t \leq T_M \\ \frac{c\gamma^{(T_M)} - c}{\ln \gamma} + \frac{t - T_M}{C_Q} & \text{for} \quad T_M < t < T_Q \\ n & \text{for} \quad t \geq T_Q \end{cases} \tag{2.28}$$

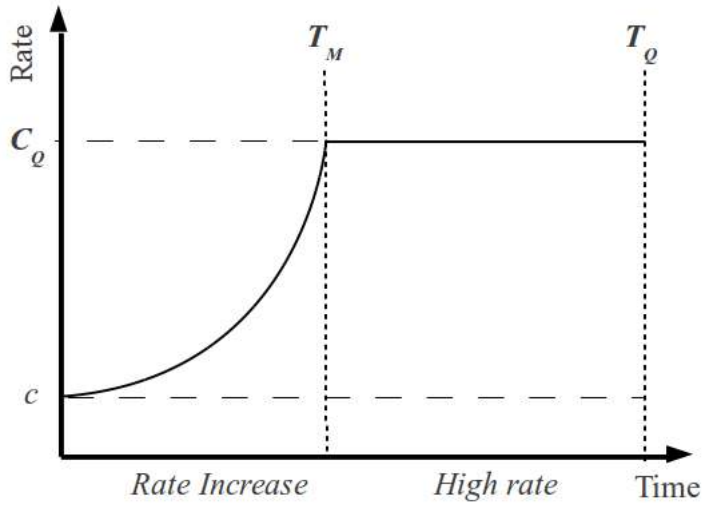



Figure 2.13: Exponentially Increasing Rate algorithm. The exponent part of the algorithm is based on a function in the form of $c \cdot \gamma^t$, where $c$ is a low transmission rate, such as $\frac{1}{RTT}$ and $C_Q$ is the maximal rate. The exponent is used while $t \leq T_M$, $T_M = \log_\gamma \left(\frac{C_Q}{c}\right)$. $T_Q$ is the deadline for all $n$ retransmissions.

### 2.4.5 Burst Linear and Burst Exponential Algorithms

We have designed two additional algorithms: "burst linear" and "burst exponential." These two algorithms are the same as Linearly Increasing Rate and Exponentially Increasing Rate respectively with one difference. Their $SP$ function gives a larger number of packets that were supposed to be sent by the original algorithm's, whilst the time spacing between transmissions, i.e. $NP$, functions are the same. This makes the algorithm "burst" packets (see Algorithm 2), before their transmission time in the original algorithm. The burst size grows along with the rate function. Note that no more than $n$ packets are sent in any case, as Algorithm 1 prevents that from happening.

In the "burst linearly" we changed $SP$ to return $t(m * t + c)$ instead of $t(\frac{m}{2}t + c)$.

Hence:

$$SP = \min\left(n, \begin{cases} t\,(m \cdot t + c) & \text{for} \quad t \leq T_M \\ T_M\,(m \cdot T_M + c) + \frac{t - T_M}{C_Q} & \text{for} \quad T_M < t < T_Q \\ n & \text{for} \quad t \geq T_Q \end{cases}\right) \tag{2.29}$$

In "burst exponential" we changed $SP$ to return $\frac{2*(c\gamma^t - c)}{\ln \gamma}$ instead of $\frac{c\gamma^t - c}{\ln \gamma}$. Hence:

$$SP = \min\left(n, \begin{cases} \frac{2\left(c\gamma^t - c\right)}{\ln \gamma} & \text{for} \quad t \leq T_M \\ \frac{2\left(c\gamma^{T_M} - c\right)}{\ln \gamma} + \frac{t - T_M}{C_Q} & \text{for} \quad T_M < t < T_Q \\ n & \text{for} \quad t \geq T_Q \end{cases}\right) \tag{2.30}$$

## 2.5 Experimental Evaluation

We have implemented an initial version of QoSoDoS and used it to test QoSoDoS' behavior and performance, as well as to compare between different ART algorithms. The implementation included Algorithms 1 and Algorithm 2 as well as all the ART-algorithms described in Section 2.4. To each packet we added a 12 bytes header, which include 3 integers: operation (SEND/ACK), message ID and retransmission number. This way we could know which retransmissions were accepted by the server and whether more than one copy of the same message was received.

Mirkovic et al. [58] discuss ways to test DoS defenses. In the paper, the authors compare between computational simulations, testbed emulations and deployment-based testing. They conclude that testbed experiments serve as the preferred way to test DoS defenses. In a nutshell, complete systems complexities and unexpected behavior is very difficult to fully capture using simulations or theoretical analysis. On the other hand, using real-world deployment-based testing is difficult, if not completely impossible. Consequently, as testbeds use real systems which are made out of actual hardware and software, they provide many of the complexities and unexpected behavior one might expect to find in a deployed system. The downside of testbeds is their (relatively) small-scale and their lack of ability to fully emulate the real-world traffic (legitimate and attack) and

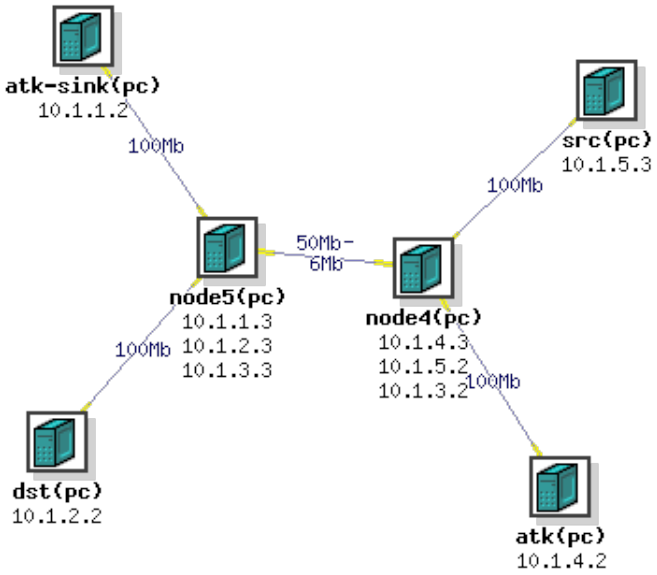



Figure 2.14: Dumbbell topology for ART comparison in DETERlab [16]

equipment heterogeneity. Nevertheless, they provide a reasonable compromise between the (unachievable) real-world and the theoretic results of simulations and analysis. We follow these recommendation and use the DETERlab testbed [16] as the infrastructure of our evaluations.

### 2.5.1 ART Algorithms Comparison

We have constructed a dumbbell topology as depicted in Figure 2.14 to compare between the different ART algorithms. All the traffic flows via the bottleneck link between *node*4 and *node*5. Attack traffic flows between the attacker and the attacker sink, denoted *atk* and *atk-sink* respectively, and the legitimate traffic flows between the legitimate QoSoDoS client and its destination, denoted *src* and *dst* respectively.

The topology consists of Linux (Ubuntu 10.04 or 12.04) machines for end-hosts, and FreeBSD for routers (*node*4 and *node*5). The attack traffic produced UDP traffic at a constant 100Mbps rate, using `iperf` [1]. The configuration used for comparing between the ART algorithms is described in Table 2.4.

| Parameter | | Description | Value |
|---|---|---|---|
| Input | $L_D$ | Network max latency | 75 ms |
| | $T_T$ | TCP (user) timeout | 6.925 sec |
| | $C_Q$ | Client max rate | 10 Mbps |
| | $P_D$ | Min bound on packet delivery prob. | 2% |
| | $P_Q$ | QoS delivery probability | 99.9% |
| | $B_Q$ | QoSoDoS assured burst | 100 Mbit |
| | $\alpha$ | QoSoDoS Relaxation Parameter | 750 |
| Calculated | $L_Q$ | Assured latency | 7 sec |
| | $n$ | Required number of retransmissions | 342 |
| | $R_Q$ | QoSoDoS assured rate | 39 bps |
| | $T_Q$ | QoSoDoS packet timeout | 63.2 sec |

Table 2.4: ART algorithms configuration.

To achieve different delivery probabilities we changed the bottleneck link capacity. The link capacity started from 50Mbps and down to 6.25Mbps (in log scale), yielding 50% down to 6.25% of the attacker's bandwidth. The ratio between the attacker's bandwidth and the bottleneck link $\min\left(1, \frac{R_N}{R_A}\right)$, is denoted $P_{N/A}$, i.e., the attacker induced delivery probability. The effective delivery probability, $P_E$, depicted in Figure 2.15, changes depending on the ART algorithm used, which also influences the delivery probability. Note that the bursty algorithms, bulk-at-start and burst-exponential, can be helpful for low delivery probabilities as their negative impact on link load is reduced, compared to the attacker rate. However, in high delivery probabilities, burstiness has negative effect on the network, and may decrease the delivery probability as numerous redundant packets are transmitted whilst only a fraction of which would suffice, and the average load produced by the bursty algorithms is relatively high.

**Latency and rate results.** We compared the latency and rate of the different ART algorithms, as depicted in Figures 2.16 and 2.17 respectively. Observe that the latency of the bulk-at-start algorithm is significantly lower than the rest of the ART algorithms. This is not surprising by itself, however, when examining Figure 2.17 we can see that the rate is comparable with the other ART algorithms. This phenomenon can be explained based on ART's ability to identify packet acceptance only after an RTT, which implies that the algorithm will keep sending for at least an RTT, until stopping, which does

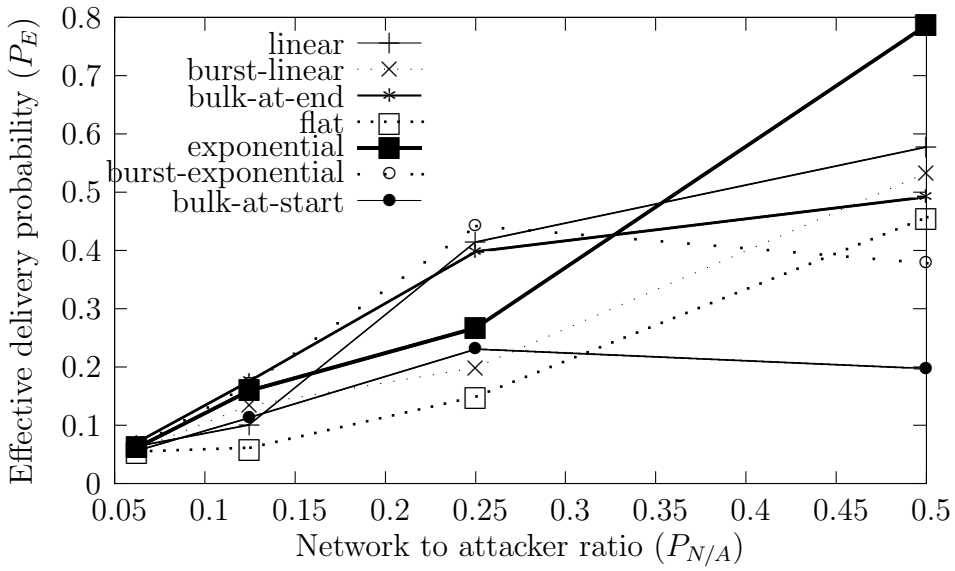



Figure 2.15: Measured $P_E$ vs. $P_{N/A}$.

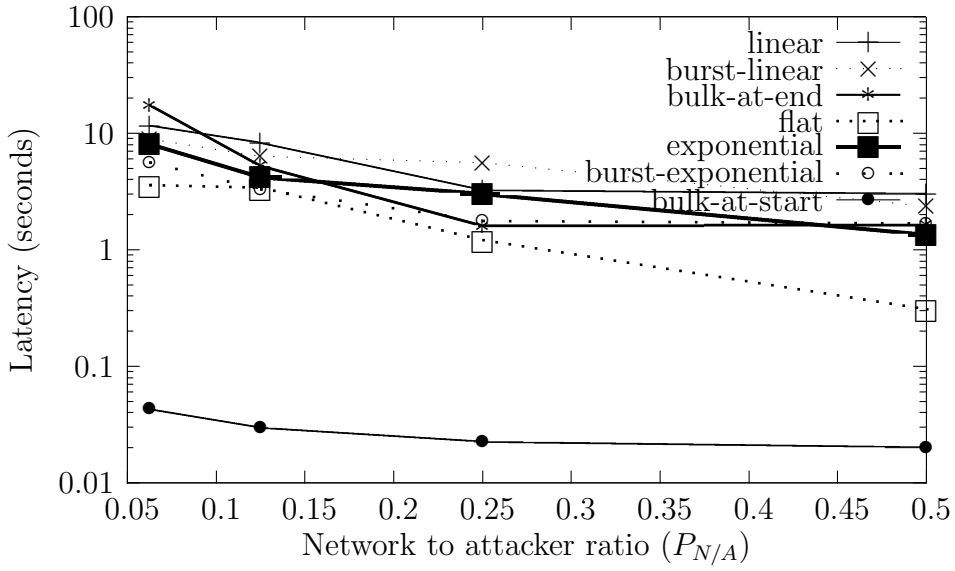



Figure 2.16: Measured latency (log scale) vs. $P_{N/A}$.

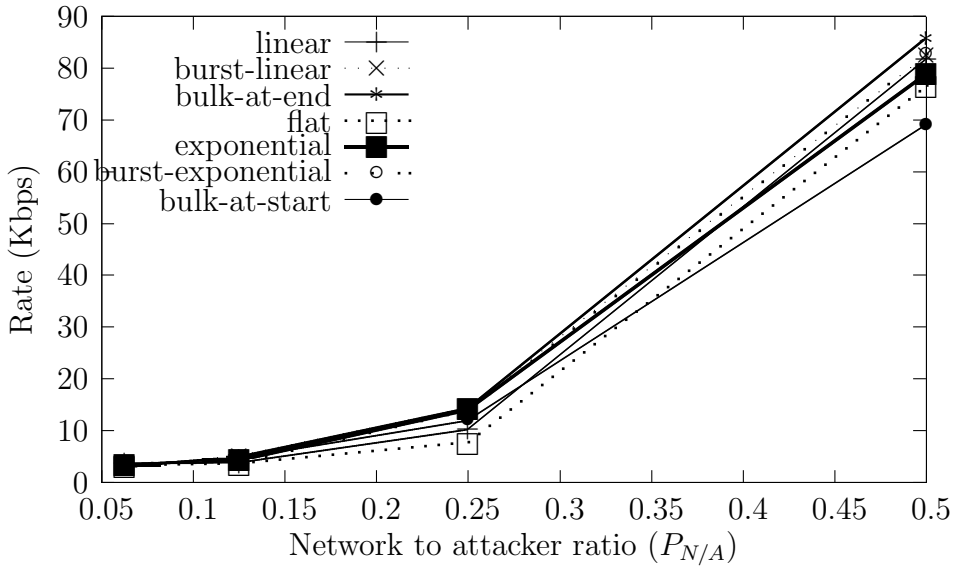



Figure 2.17: Measured rate vs. $P_{N/A}$.

not allow it to continue for the next packet, even though one of its packets has been successfully accepted. This observation is supported by Figure 2.18, in which we see the significant amount of redundant packets sent (and received) by bulk-at-start.

**Redundant packets results.** Another observation regarding Figure 2.18 is that lower delivery probabilities cause *more* redundant packets to be accepted. The reason for this is that while we have changed the link's bandwidth, we did not change its queue size, which implies that the RTT increases as the same number of packets in the queue are being delivered using a lower outgoing rate. Hence, the time until acknowledgement is accepted is prolonged. During this time, the bulk-at-start algorithm sends more packets, and, as depicted, more redundant packets are effectively accepted.

Figure 2.18 depicts the number of packets redundantly received by the destination server. Note that in the configuration used (see Table 2.4), only the bursty algorithms exhibited such redundant behavior, as shown in the figure.

### 2.5.2 Mixed Senders and Attackers Tree

In this experiment we evaluate transmissions by nodes which send a mix of senders and attackers. This emulates a scenario where there are two congested locations. The first is

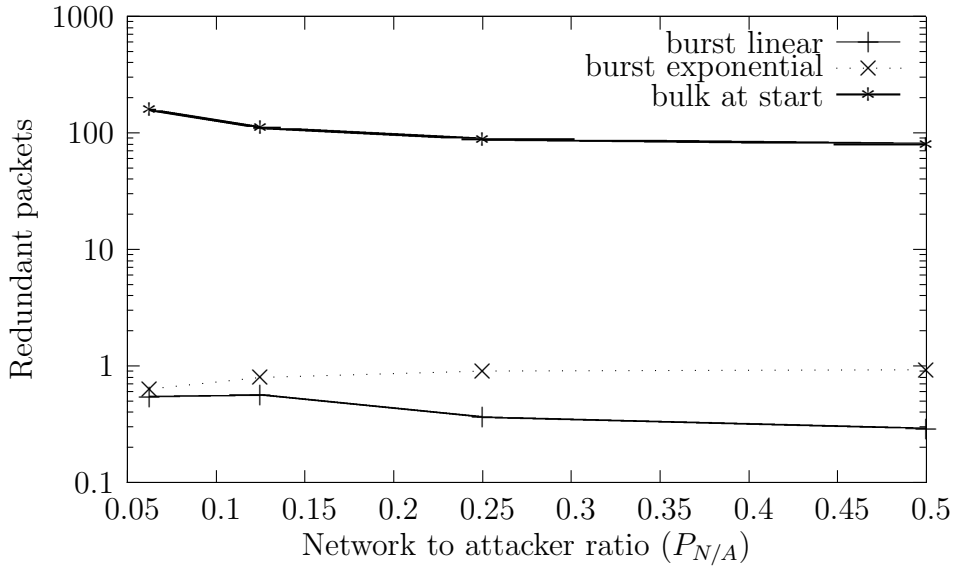



Figure 2.18: Measured Redundant Packets vs. $P_{N/A}$.

at the last-mile bottleneck link, as before. The second is another upstream bottleneck, closer to the sources. Such a scenario may happen if an attacker cohabits the same machine or the same network as legitimate clients, or otherwise when another bottleneck is encountered en route. In this scenario the congestion on the last-mile bottleneck link does not provide the full picture regarding delivery probability, as some packets are already lost before reaching the last-mile bottleneck link by some congested upstream machine.

We have set up a network in a tree topology as depicted in Figure 2.19, in which we emulated various attack sizes and compared QoSoDoS with TCP. The network topology is constructed so that all transmissions towards the destination host $D$ congest the same link.

All the machines, including routers, are Linux machines with kernel version $\geq$ 2.6.18. Both clients and server used Ubuntu 10.04 with the default TCP congestion control (CUBIC [69]). Each machine has 100Mbps Ethernet NIC(s). $R_0$ and $R_1$ are routers with three and two Ethernet interfaces respectively. $S_1$ and $S_2$ connect to the first NIC of $R_0$, whereas $S_3$ and $S_4$ connect to the seconds. The third NIC of $R_0$ is connected to the first NIC of $R_1$. The second NIC of $R_1$ is connected to $D$ using a rate-limiter rating it to

10Mbps with buffer size of 750Kbps.

Each source node consists of several QoSoDoS clients and at least twice as many attackers with higher scheduling priority, transmitting as many packets as required to consume enough bandwidth according to the attack scenario. Each client transmits packets with a payload of 262-bytes which emulates a 12-bytes QoSoDoS header and 250 bytes of payload. For this experiment, attackers transmit a 12-bytes payload over UDP, hence producing a minimal Ethernet packet with a size of 64-bytes. Such a packet size is typical to TCP packets carrying no payload, such as SYN, ACK, FIN and RST. Motivation for having such a small attacker packet size is to give the attacker's packets better queuing probability in routers.

For all experiments described in this section we used the *linear* ART algorithm.

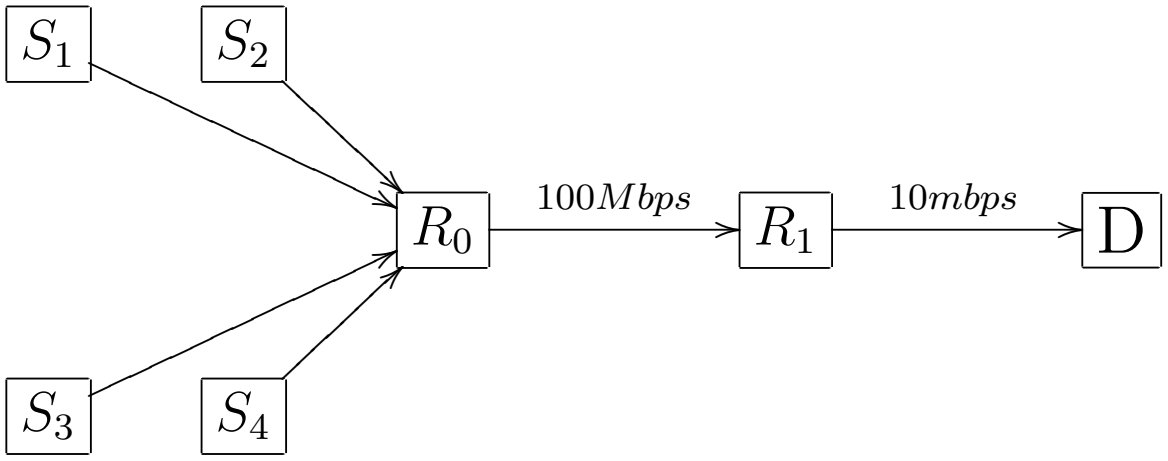



Figure 2.19: Mixed senders and attackers experiment. $S_1..S_4$ are sources of packets containing both QoSoDoS clients and attackers. Each QoSoDoS client is configured to use no more than 100Kbps when executing QoSoDoS-ART scheme (see Table 2.5). The amount of packets and bandwidth created by the attackers changes to emulate various probabilities for QoSoDoS' successful packets delivery. $R_0$ is a router with a 100Mbps rate connected to router $R_1$ which limits the rate towards $D$ to 10Mbps. Each single source does not transmit any more than 40Mbps so that the underlying Ethernet links won't affect the results.

### 2.5.3 Various Attack Sizes

On each source machine, $S_1..S_4$, we have executed six concurrent QoSoDoS clients with parameters as described in Table 2.5. In addition, we have executed 14 attackers on each

| Parameter | | Description | Value |
|---|---|---|---|
| Input | $L_D$ | Network max latency | 75ms |
| | $T_T$ | TCP (user) timeout | 2.925s |
| | $C_Q$ | Client max rate | 100Kbps |
| | $P_D$ | Min bound on packet delivery prob. | 0.1% |
| | $P_Q$ | QoS delivery probability | 99.9% |
| | $B_Q$ | QoSoDoS assured burst | 10Mbits |
| | $\alpha$ | QoSoDoS Relaxation Parameter | 1.5 |
| Calculated | $L_Q$ | Assured latency | 3s |
| | $n$ | Required number of retransmissions | 6905 |
| | $R_Q$ | QoSoDoS assured rate | 9.65 bps |
| | $T_Q$ | QoSoDoS packet timeout | 255.2s |

Table 2.5: QoSoDoS' parameters used in experiments.

machine which produce the bandwidth-flooding of the link.

Each experiment was executed as follows. 30 seconds after the clients were all executed, an attack was launched for 30 minutes, followed by 60 seconds for clients recovery and testing whether TCP was resumed properly. For all attack sizes we examined, all the QoSoDoS clients resumed TCP almost immediately.

We have tested attack sizes ranging from 10Mbps to 150Mbps, which provided an effective acceptance probability ($P_E$) ranging from 4.5% to 0.3% as described in Figure 2.22.

Figure 2.20 and 2.21 present the average latency and rate (respectively) vs. packet acceptance probability. As expected, packets were accepted at lower latencies and higher rates than assured by QoSoDoS, since the packets' expected number of retransmissions until delivery is $\frac{1}{P_E}$.

### 2.5.4 QoSoDoS and TCP Comparison

We conducted two experiments to compare TCP and QoSoDoS. In the first experiment we tested the goodput of TCP vs. the goodput of QoSoDoS. We executed the same number of clients for 5 minutes and tested how much data was delivered using TCP and how much data was delivered using QoSoDoS. For each QoSoDoS client we used the

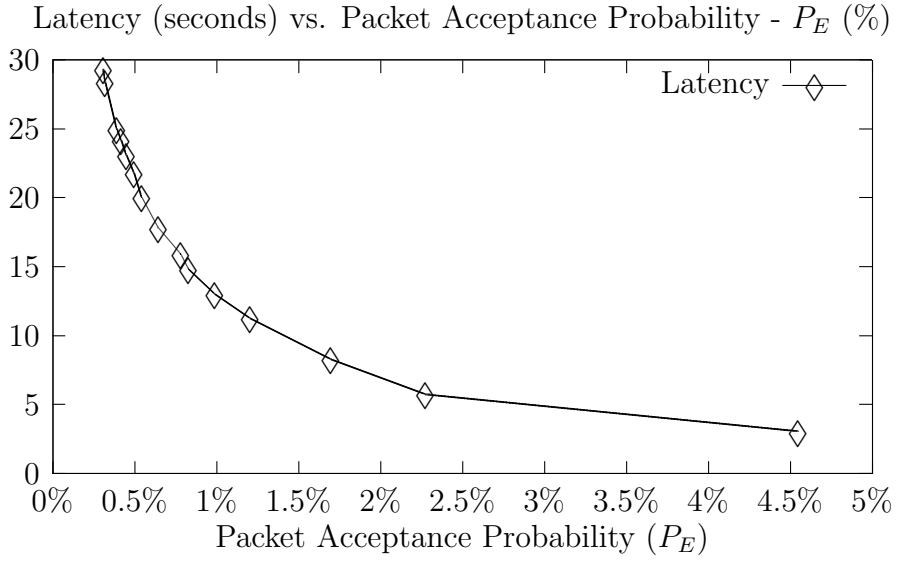



Figure 2.20: Latency (seconds) vs. acceptance probability ($P_E$). The effective latency ranges between 29.3 and 3 seconds. Note that the latency values are lower than the assured value $T_Q$ (see Table 2.5), as the average packet is accepted by the mean value of the effective probability ($\frac{1}{P_E}$), producing *much lower* latencies than the assured worst case.

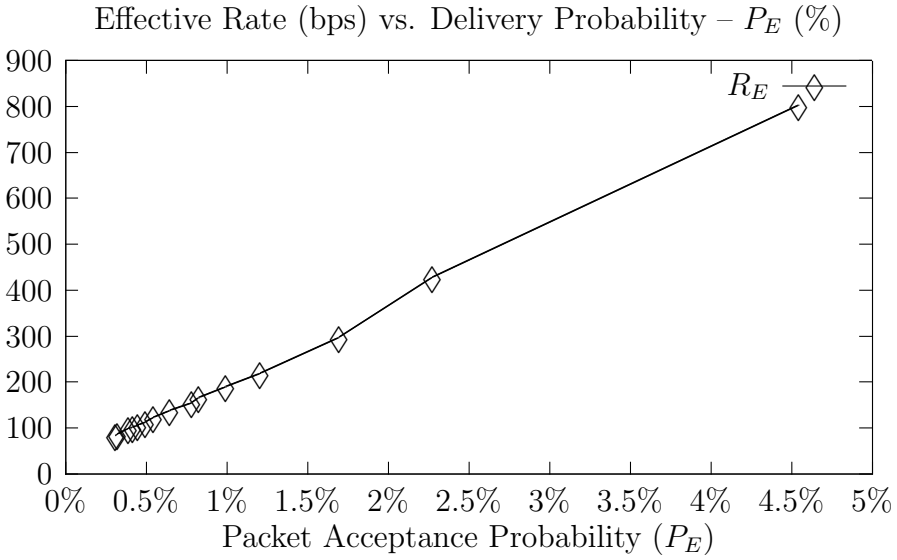



Figure 2.21: Effective Rate vs. Delivery Probability ($P_E$). The effective rate ranges between 803 bps and 84 bps. The rate values are much higher than the assured $R_Q$ (see Table 2.5), as the average packet is accepted by the mean value of the effective probability ($\frac{1}{P_E}$), producing *much higher* rates than the assured worst case.

same configuration as described in Table 2.5. Figure 2.23 shows that using up to 100 concurrent clients, QoSoDoS performs within 2.5% of TCP, which seems like a reasonable

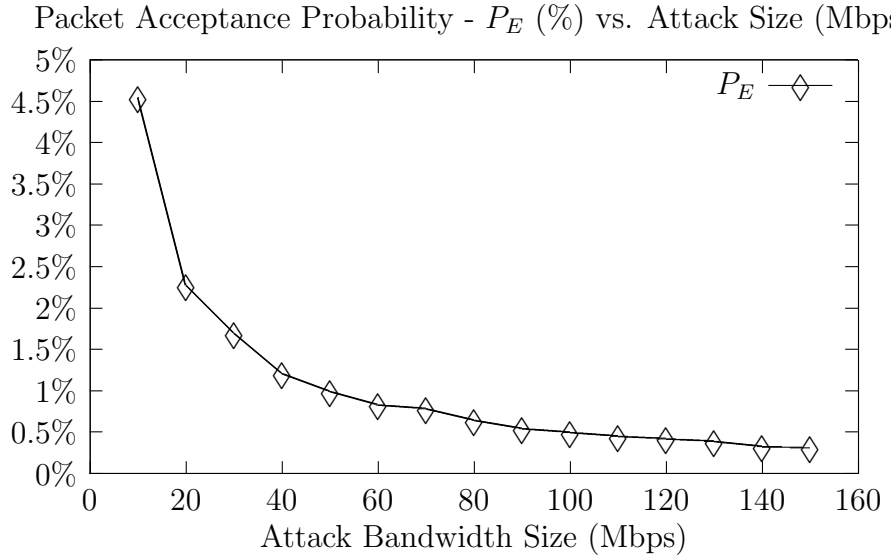



Figure 2.22: Effective packet acceptance prob. ($P_E$) vs. attacker's bandwidth. Even for a strong attacker ($\times$15 link bandwidth), the acceptance probability is 0.3%. Note that the delivery probability is dependent on more than one bottleneck, both at the source machines as well as the bottleneck link.

price to pay for the assured QoS. Figure 2.24 shows that the amount of QoSoDoS packets using ART (rather than TCP) is negligible; less than 0.12% (about one promil) of the accepted packets were ART packets when 100 QoSoDoS clients were executed concurrently. This demonstrates that QoSoDoS does not self-create DoS even when many clients run concurrently.

The second experiment we conducted to compare QoSoDoS and TCP included a short-lived attack. Like the previous experiment, we started without an attack. After 60 seconds of running TCP, we launched a 10Mbps attack ($P_E = 4.5\%$) which lasted 40 seconds. After the attack was over, we continued running the experiment for an additional 200 seconds (2.3 minutes). This kind of attack is intended to simulate a flash crowd, or an attacker with low and/or short-lived capabilities. Low-rate attacks [49] can be regarded as an extreme case for this type of an attack, which are very hard to detect and filter. In this case, *QoSoDoS performs better than TCP*. Figure 2.25 shows *TCP degradation* compared with QoSoDoS. In a small number of clients, QoSoDoS performs 11.5% better than TCP. This is reduced to about 9% when the number of clients is increased to 100.

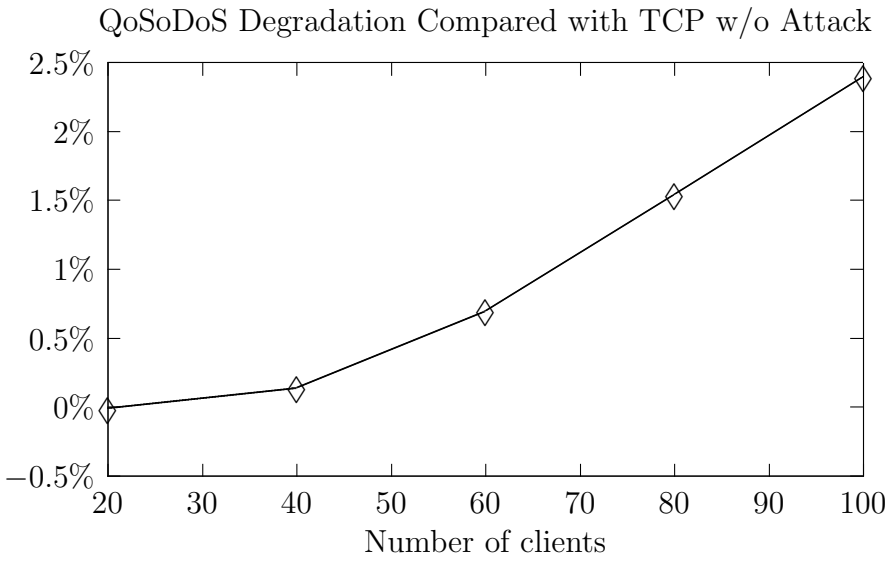



Figure 2.23: QoSoDoS degradation compared with TCP. The X-axis is the number of concurrent clients and the Y-axis is the performance degradation ranging between 0 and 2.5%. QoSoDoS performs quite well even when no attack is launched. This seems like a reasonable price for assured QoS during flooding DoS attacks.

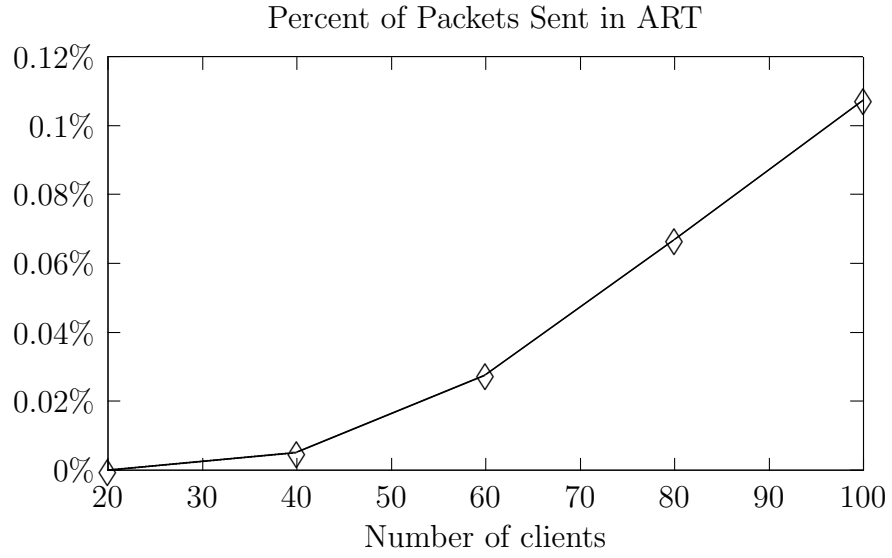



Figure 2.24: Percent of packets sent in ART. The X-axis shows the number of concurrent clients and the Y-axis shows the percentage of packets accepted using ART. The values range between 0 and 0.12%, i.e. ART overhead is negligible. Note that this means that QoSoDoS has almost no overhead, adds only a negligible amount of non-TCP traffic, and does not become a source of congestion, let alone self-created DoS.

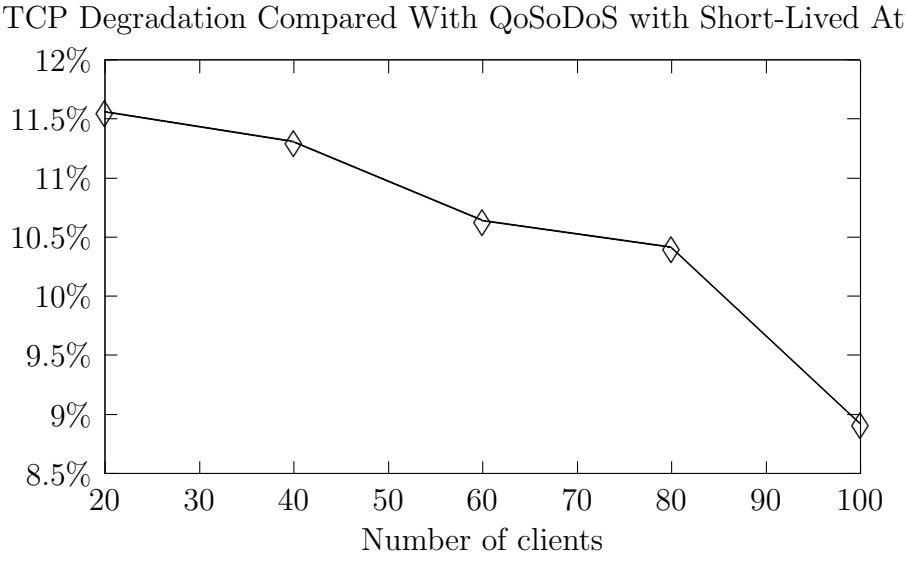



Figure 2.25: TCP performance degradation compared with QoSoDoS, under a short-lived attack during a connection. *TCP performs worse than QoSoDoS* in the range of 11.5% to 9%. We believe that this phenomenon is due to QoSoDoS' ability to recover faster than TCP from DoS attacks, which is mainly the result of its TCP timeout mechanism $T_T$.

Figure 2.26 shows that the overhead of ART packets remains almost as small as without an attack (about one promil). These results suggest that QoSoDoS recovers very well from such small-sized and/or short-lived attacks which target TCP timeouts and recovery time. We believe that the main reason behind these results is mainly due to TCP timeout – $T_T$ – which helps fast recovery when an attack is over. The results presented in Figure 2.26 support our claim that even after a DoS attack, QoSoDoS does not self-create DoS.

### 2.5.5 Link Failure Experiment

The experimental results presented here were produced using the circle-topology as depicted in Figure 2.27, using the DETERlab testbed [16]. In this experiment 10 QoSoDoS clients were executed on each node with the configuration described in Table 2.6. LAN links are 1Gbps and WAN links (connecting the routers) are 100Mbps. The routing tables were configured statically such that every packet to the server travels clockwise and every packet from the server travels counter-clockwise. For example, packets traveling from *lan*-0 (*node*-[0..2]) to *serverlan*-2 are routed via *router*-1, in addition to packets

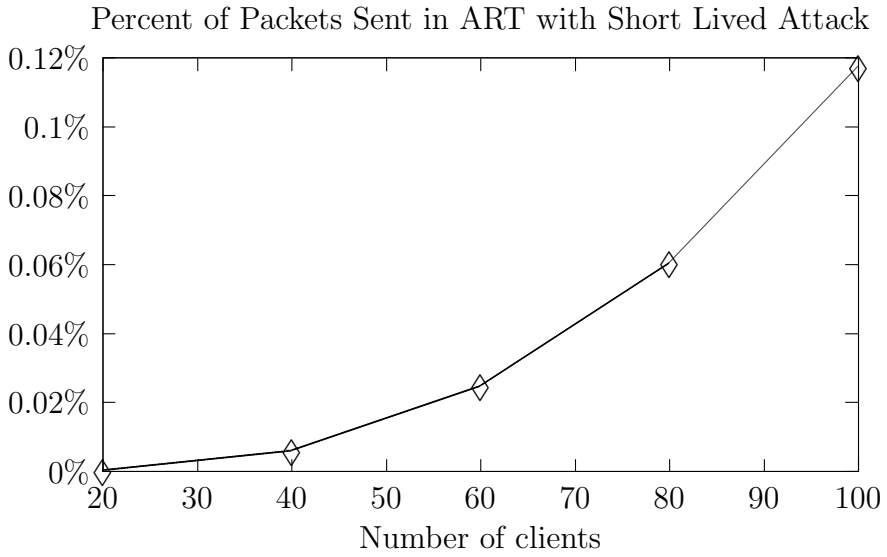



Figure 2.26: Percent of packets sent in ART with a short lived-attack. Examining the percent of ART packets in the context of a short-lived attack yields that ART presents almost no overhead. This is comparable with the result in Figure 2.24. This result further supports the claim that QoSoDoS does not become a source of self-created DoS.

traveling from *serverlan*-2 to $lan - 0$. Such a circle topology is prone to congestion explosion in case congestion is not controlled well enough. Note that $C_Q$ is configured to 100Mbps for *each* client, which theoretically provides much higher bandwidth usage than the WAN links' bandwidth, hence it can potentially produce DoS.

The experiment scenario was to execute clients for five minutes, and after one minute of execution to drop the link between *router*-1 and *router*-2. The experiment was executed once with TCP-only clients and then with QoSoDoS clients using the configuration described in Table 2.6. The results of this experiment showed that QoSoDoS sent an average of about 10-12% more packets using TCP. The second interesting result was that the average bandwidth used in total for QoSoDoS was about 120Mbit (15MB) in total, for all ART transmissions. We believe that the reason for QoSoDoS' better performance is its better recovery from link outage. After a link outage TCP enters congestion avoidance, while QoSoDoS starts a new connection with slow-start, which grows the congestion window faster.

The second experiment executed the same basic scenario with 1% packet loss on each

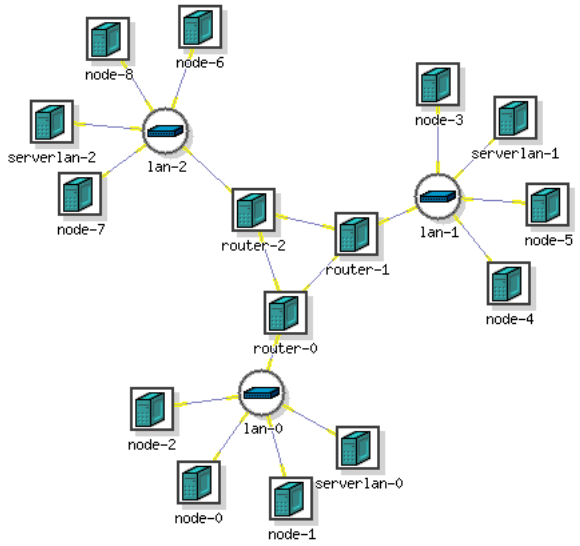



Figure 2.27: Circle topology emulated in DETERlab. LAN links have 1Gbps of bandwidth, while links between routers have 100Mbps. Packets from client nodes to servers are (statically) sent clockwise, while server responses are sent counter clockwise (e.g. *node*-0 $\rightarrow$ *serverlan*-2 is sent via *router*-1 and the response follows the same path in the opposite direction).

link. Results here show that both TCP and QoSoDoS performed the same. ART overhead was around 90Mbit, which was produced during the link down period. This result can be explained by the TCP congestion control mechanism, which was forced to use less bandwidth, due to the packet loss. In the scenario without packet loss, when many TCP clients congest the link and push it to its limits, an occasional TCP connection exceeds the 3 seconds $T_T$ timeout, hence causing an occasional packet transmission in ART (which also increases QoSoDoS' overhead).

## 2.6 Conclusions

In this chapter we have presented QoSoDoS, an end-to-end transport protocol assuring QoS over DoS-prone networks. We compared between different ART transmission schemes. Experimental results show that QoSoDoS has good performace during large-scale BW-DDoS attacks. Compared to TCP under normal conditions QoSoDoS presents

| Parameter | | Description | Value |
|---|---|---|---|
| Input | $L_D$ | Network max latency | 75 ms |
| | $T_T$ | TCP (user) timeout | 2.925 sec |
| | $C_Q$ | Client max rate | 100Mbps |
| | $P_D$ | Min bound on packet delivery prob. | 0.1% |
| | $P_Q$ | QoS delivery probability | 99.9% |
| | $B_Q$ | QoSoDoS assured burst | 100 Mbit |
| | $\alpha$ | Relaxation parameter | 1.5 |
| | $\phi_{init}$ | Faulty state initialization parameter | 3 |
| | $\phi_{sample}$ | Faulty state sampling parameter | 5 |
| Calculated | $L_Q$ | Assured latency | 3 sec |
| | $n$ | Required number of retransmissions | 6905 |
| | $R_Q$ | QoS (Assured) rate | 14.5Kbps |
| | $T_Q$ | QoSoDoS message timeout | 255ms |

Table 2.6: QoSoDoS' parameters used in the link failure experiment.

little overhead, whilst during and after an attack its performance is superior. QoSoDoS does not self-create DoS attacks and recovers well after such attacks.

# Chapter 3

# Controlled Overlays

In this chapter we investigate the effectiveness of using overlay networks to break through congested networks as a result of bandwidth flooding, such as DDoS attacks (BW-DDoS). We present a novel design which, upon congestion, turns to an end-host-based overlay to redirect communication and amplify legitimate traffic, using one or multiple paths to the destination as necessary. Our overlay is carefully controlled, thus preventing self-generated DDoS.

We have evaluated our proposed overlay scheme using a large-scale simulation, based on the Internet's AS topology. We examined the effectiveness and overhead of our design as well as that of TCP-based relaying schemes, such as RON [9]. Our results show considerable improvement in delivery probabilities and acceptable overhead. To our knowledge, this is the first large-scale evaluation of the use of overlays to ensure availability in spite of a significant BW-DDoS.

## 3.1 Introduction

The Internet provides a best-effort packet delivery service, with routing mechanisms which respond to congestion only slowly, if at all. In particular, inter-AS routing by BGP does not consider congestion at all, neither inter-AS nor intra-AS. Most providers

ensure sufficient overcapacity, to avoid such congestion along the route from source to destination AS, however, this may fail against a determined DDoS attack such as the large attacks recently reported [12].

In recent years, there is a growing number of incidents of large-scale, well-organized DDoS attacks with significant attacker resources [12,30,41,42]. Recent DDoS attacks [39] have shown that strong attackers can produce attacks at an alarming rate of millions of packets per second, yielding a few dozens of Gbps. Such large-scale attacks can cripple very large sites, as well as various mitigation systems which cannot handle the overwhelming magnitude of the attack's bandwidth.

In practice, many solutions can be considered to mitigate DDoS. A common solution is over-provisioning the potential victim with bandwidth and other resources, making it harder to launch an effective DDoS attack. However, resource over-provisioning is costly, and cheaper solutions are widely available, such as attack flows filtering by ISPs, or spawning new point-of-presence (PoP). We further discuss two types of off-the-shelf solutions, namely, in premise, such as Corero DDS [40], and "in the cloud" solutions, such as Akamai DDoS Defender [38]. Customer-premises equipment (CPE) solutions assume that the victim has sufficient bandwidth but cannot handle the attack at application level. Hence, traffic can be filtered at the victim's premises before it exhausts the victim servers' resources. In the cloud solutions are based on diverting the traffic to a cloud service which can absorb the attack bandwidth and filter out the offending flows. not under attack, or ad-hoc upon attack.

DDoS attacks have also motivated vast research in various aspects of DDoS mitigation, including the use of *overlay networks*, i.e., the use of helper nodes as intermediaries between source and destination, to ensure availability even when under severe DDoS attacks. Overlay networks research can generally be divided into two types: *Indirect Overlay Networks* (ION) and *absorption overlays*.

IONs try to improve end-to-end availability and performance by detouring the Internet's core routing, and overcoming BGP's shortcomings, such as speed of update, route

selection using different matrices, or using special network features such as multihoming; e.g. see RON [9],Bandwidth-Aware Routing in Overlay Networks [50] and MONET [10]. IONs can mitigate DDoS in cases where some routes to the victim are congested, while others are not congested and are able to deliver data to the victim. However, once all routes to the victim are congested, existing IONs will most probably prove futile.

*Absorption overlays* absorb the attack bandwidth at overlay nodes, and only forward legitimate traffic to the destination. Absorption overlays were proposed and investigated in several works, such as *SOS* [45], *Mayday* [25], *Phalanx* [27], and *Countering DoS Attacks With Stateless Multipath Overlays [72]. Such overlays commonly construct a* secure perimeter *around the victim, through which only a selected set of authorized nodes can deliver data. Authorized nodes are commonly authenticated using lightweight authenticators, e.g. IP, port etc. On the other hand, even for absorption overlays, the perimeter itself has limited bandwidth capacity. Thus, attackers may try to clog the perimeter borders, and severely limit the connectivity between the overlay and the secure perimeter. Ultimately, existing mitigation techniques have some bandwidth limitations, whether it is in routers, firewalls, secure perimeters, or even a highly provisioned link.*

*In this work, we study the use of overlay networks to increase availability, in the presence of severe bandwidth flooding, possibly by an intentional DDoS attack. We take a* complementary approach *to existing solutions, both overlay-based and others, to fight such bandwidth floods. We accomplish this by amplifying bandwidth and aggressively delivering the legitimate traffic in spite of existing congestion. The key feature is that, when necessary, we let a set of* legitimate and controlled overlay nodes *collaborate and transmit data, at a predefined (controlled) rate, but without congestion control, i.e., using UDP instead of TCP. We thereby assure information delivery to the victim host, even when it is under a large-scale flooding attack. Since the set of overlay nodes is controlled, we can control the worst-case bandwidth usage, and avoid self-generating DDoS. Employing our solution will require a much stronger attacker utilizing a much stronger botnet to achieve the same effect as it can today, and it requires no change to the Internet's core or to*

*routers.*

*Figure 1.2 depicts an Internet-scale simulation which motivates our solution. The black bars depict ASes, fully available under various scales of bandwidth attacks; gray bars depict partially available ASes to which IONs may offer some solution. The white bars depict the area of inaccessible ASes by existing solutions, to which, in conjunction with the gray bars area, we offer a solution in this chapter. Note that the white area in Figure 1.2 is also relevant for protecting the secure perimeter of absorption overlays.*

*Our solution should be less costly than statically over-provisioning resources, and should be complementary to existing solutions. It should be easy to deploy for example by maintaining online overlay relays or spawning cloud relay instances on demand. Despite the bandwidth costs, many services have considerable costs from going offline, hence our solution should be cost-effective for such services. Our solution is strictly designed for cases in which we can differentiate friend from foe, i.e. when clients can be authenticated by the overlay and banned for misbehaving, e.g. banking and emergency services. Our solution is inappropriate for open services which do not require any client authentication, in which an attacker might try misusing the overlay for its own attack amplification.*

*The rest of the chapter is organized as follows. Section 3.2 presents the model assumptions. Section 3.3 describes the design followed by the evaluation found in Section 3.4. We conclude in Section 3.5.*

## 3.2 Model Assumptions

### 3.2.1 Network Behavior

*We take the delivery assumptions as described in Section 1.1.1. Specifically we assume that the delivery probability $P_D$ is approximated by Equation 1.1, i.e., $P_D \approx \min\left(1,\ \frac{R_O}{R_I}\right)$.*

*Next, as depicted in Figure 1.1, traffic without congestion control is "prioritized" over congestion controlled traffic, i.e., TCP lets UDP flows transmit their bandwidth and use only the remaining bandwidth.*

### 3.2.2 Bandwidth Costs

*We try to estimate the costs associated with an aggressive relaying scheme, and conclude whether our scheme is financially reasonable. Chen et al. [21] calculated cloud usage costs, including networking costs to and from the cloud. In this chapter we use these assessments to estimate the costs of our relaying scheme. We further focus on bandwidth costs, as processing costs are negligible compared to bandwidth costs. Assuming that our relays are cloud instances, and using worst-case pricing, i.e.* $4,500$ *pico* ($10^{-14}$) *cent per bit (pcb) for end-host transmissions, in addition to* $1,164$*pcb for the cloud's data-in, and* $1,979$*pcb for the cloud's data-out, we get a total of* $7,643$*pcb* for a single end-to-end bit relaying. *In our design, relaying through the cloud should be retransmitted several times by the cloud, hence reducing the costs, as the end-host transmission plus cloud data-in are significantly larger than just the cloud's data-out. Even so, transmitting 1 megabit using 100* end-to-end relaying – *an example of the expected number of retransmissions for* $P_D = 1/100$ – *we get* less than 0.77 cent per transmission *(comparable to a single SMS price). Since commonly attacks don't get to such low delivery probabilities and since the amount of data to deliver can be smaller than 1 Mbit, then based on these costs we argue that there exist a group of applications for which our solution is cost-effective. Additionally, some types of transmissions are worth significantly more than one cent, such as emergency services messaging or financial transactions, which make our proposed cloud-based scheme financially reasonable at least for similar types of applications.*

## 3.3 Design

*Our basic design is based on using standard TCP over the Internet whenever enough bandwidth is available for TCP streams. However, whenever the network becomes too congested, we do two things. First, we begin transmitting using UDP, hence exploiting the real delivery probability. Second, we begin using an overlay to relay data to the destination in parallel, as described in Section 3.3.5.*

### 3.3.1 Cloud-based Overlay

*We suggest that operators of potential victim servers deploy an overlay across the Internet at different locations, making attacks on the overlay itself impractical. We assume that most of the time the server is not attacked, hence ad-hoc cloud services can be instantiated only upon attack. The operator can instantiate just enough nodes, constantly considering links' limitations. To prevent attackers from misusing the overlay for amplification, overlay nodes should require client authentication, e.g. by using TLS client authentication, proof-of-work such as SpeakUp [79], CAPTCHA [6] or other previously proposed means for DoS mitigation in overlays [25,27,45]. Our solution is inappropriate for open services which are unable to differentiate attackers from legitimate clients.*

### 3.3.2 Clients-based Overlay

*A second scheme for building an overlay is by using the server's clients. Most servers have many clients, some of which would probably be willing to help each other on a tit-for-tat basis. We can use the fact the destination server is involved and can act as a trusted bookkeeper. In this setup the server can offer premium service for cooperating clients, or even offer a payment, and in turn charge clients who are using the overlay, thus also avoiding free-riders.*

### 3.3.3 Source and Relay Authentication

*To mitigate spoofing and forgery by either sources or relays, we need to use cryptographic primitives to prevent both source and relay from launching attacks on behalf of otherwise legitimate clients. On the other hand, since the victim server is already attacked and we do not wish to add new computational DDoS attack vectors, we try to avoid using computationally expensive solutions. Therefore, we try to make sure that the server will make as few PKI operations as possible. The requirement are therefore as follows:*

1. *The destination node should be able to authenticate the relaying node of the message.*

2. *The destination node should be able to authenticate the source node of the message.*

3. *Optionally, in case of misbehavior the destination node should be able to detect which of the source and relay nodes is misbehaving.*

4. *The destination should use as few PKI operations as possible.*

*To that end, we assume that prior to the attack, the server shares a key with each of its clients, which is used for client authentication. Additionally, the server supplies the client with a server-signed certificate, containing the client's public key, and a server-signed list of nodes it may use as relays.*

*We will be using the following notation.* $\mathcal{S}_x(\cdot)$ *is a signature using a signing key of* $x$. $\mathcal{MAC}_k(\cdot)$ *is a* Message Authentication Code *using a shared key* $k$. $\mathcal{CERT}_x(n)$ *is a* Certificate*, e.g., X.509 [24], signed by* $x$*, authorizing node* $n$ *(i.e.* $n$*'s public key).*

*Figure 3.1 depicts a protocol sequence, in which prior to the attack, the destination server, denoted* $D$*, provides the source client, denoted* $S$*, and a relay, denoted* $R_i$*, where* $i$ *is the relay's ID, with signed certificates* $\mathcal{CERT}_D(S)$ *and* $\mathcal{CERT}_D(R_i)$ *respectively. Additionally, the server provides* $S$ *with a signed list of relay nodes it may use. When the client* $S$ *refers to relay* $R_i$*, it first identifies itself using the signed certificate* $\mathcal{CERT}_D(S)$*. This process can be performed, for example, using TLS [26] or IPsec [31], and it should include client and server authentication between the source client and the relay, hence preventing anyone from spoofing client messages to relays. After the authenticated channel has been established, the source client should present the relay with a destination signed* relay-list *issued to that specific source client. The destination signed relay-list should contain, at least, the source client's ID and the relay's ID (i). Until* $S$ *can provide* $R_i$ *with the destination signed relay-list,* $R_i$ *should not relay any of* $S$*' data. Once* $S$ *has a relaying permission from* $R_i$*, it can start relaying messages to* $D$ *via* $R_i$*.*

*Next, we need to enable* $D$ *to authenticate both the relay* $R_i$ *and the source* $S$*, with minimal number of PKI operations. For* $R_i$ *authentication we use* $\mathcal{MAC}$ *with pre-shared keys between* $R_i$ *and* $D$*. Since we also want to prevent* $R_i$ *from spoofing* $S$*' messages,* $S$

*will also add a $\mathcal{MAC}$ with another pre-shared key between S and D. Hence D is required to authenticate two MACs, without any computationally expensive PKI operations. The first $\mathcal{MAC}$ to be authenticated is $R_i$'s $\mathcal{MAC}$, and the second $\mathcal{MAC}$ to be authenticated is S'. If $R_i$'s authentication failed, there is no point in authenticating S' $\mathcal{MAC}$, as the entire message is probably forged.*

*In case $R_i$'s $\mathcal{MAC}$ is authenticated, but S' MAC is not, D should request $R_i$ to refrain from relaying any further messages from S. This decision is based on the fact that we do not assume that either $R_i$ or S are honest. Hence $R_i$ could have spoofed S' message, or S is delivering a malformed message or $\mathcal{MAC}$, or otherwise S or $R_i$ may be compromised. In any case, $R_i$ should stop relaying such packets. If $R_i$ continues to deliver messages from S, then $R_i$ can be banned by D for misbehaving. To prevent replay attacks of messages sent to the relay or from the relay, relayed messages should always contain a nonce from S as well as another nonce added by $R_i$.*

*$R_i$ may refrain altogether from transmitting S' data, which as far as we are concerned is acceptable, as we do not presume that any node must cooperate in relaying. Instead, nodes should be motivated to cooperate as discussed above. Finally, if an attacker can impersonate S or $R_i$, this would imply that it is able forge the certificate signed by D or forge a $\mathcal{MAC}$.*

*Using the optional PKI-based operations, shown in brackets in Figure 3.1, we get a protocol which enables the destination D to identify the malicious node, S or $R_i$. This may be followed by D banning the malicious nodes from (using) the overlay. The protocol begins similarly to the simpler protocol, i.e., prior to the attack $R_i$ and S receive from D certificates, $\mathcal{CERT}_D(R_i)$ and $\mathcal{CERT}_D(S)$ respectively, and shared keys with D, $k_r$ and $k_s$ respectively. During congestion, S establishes an authenticated channel with $R_i$ and delivers $S_D(S||\{R_i, ...\})$ to $R_i$. Next S constructs a message, denoted $M_S$, which consists of the message M to deliver, a nonce $n_s$ to prevent replays, and $\mathcal{MAC}_{k_s}(M||n_s)$. S then sends $M_S$ alongside a signature $\mathcal{S}_S(M_S)$. The relay cannot authenticate $\mathcal{MAC}_{k_S}(M||n_s)$, however it should be able to verify the signature $\mathcal{S}_S(M_S)$. If $mathcalS_S(M_S)$ is ver-*

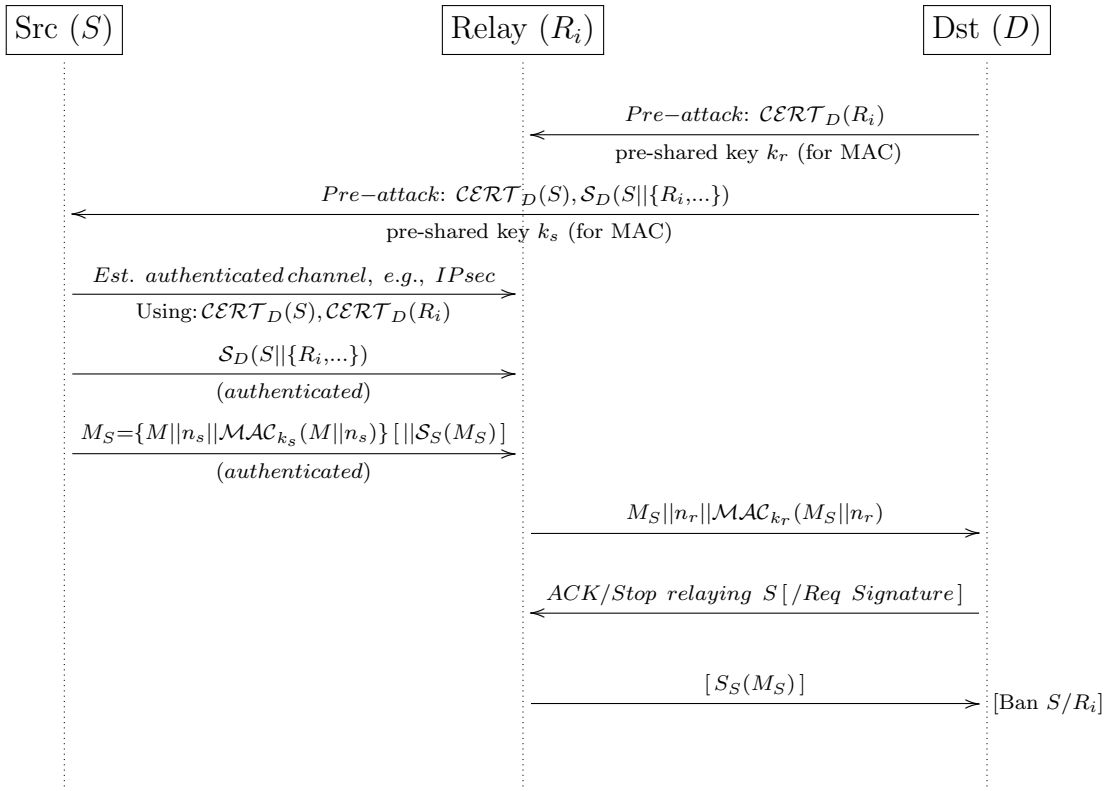



Figure 3.1: Client-based overlay protocol, with an optional extension (in brackets) for identifying malicious nodes ($S$ or $R_i$) using minimal PKI operations. $M$ is the message to deliver, $n_s$ and $n_r$ are nonces added by $S$ and $R_i$ respectively, to prevent message replay attacks. Without the extension the destination can only acknowledge receiving message $M$ or ask $R_i$ to stop relaying $S$' messages. Using the extension, the destination can specifically identify and ban the malicious node $R_i$ or $S$.

*ified, $R_i$ will concatenate $M_S$, its own nonce $n_r$ and $\mathcal{MAC}_{k_r}(M_S||n_r)$, and send it to the destination $D$. Once the message arrived at the destination, $D$ should authenticate $\mathcal{MAC}_{k_r}(M_S||n_r)$ followed by authenticating $\mathcal{MAC}_{k_s}(M||n_s)$. If both are authenticated, and no replay was detected, then $D$ can process the message.*

*In case either $\mathcal{MAC}$ was not authenticated, there are several conditions to consider:*

- *If $\mathcal{MAC}_{k_r}(M_S||n_r)$ was not authenticated, then the message might have been spoofed and can simply be ignored.*
- *If $\mathcal{MAC}_{k_r}(M_S||n_r)$ is authenticated but $\mathcal{MAC}_{k_s}(M||n_s)$ is not authenticated then $D$ should request $\mathcal{S}_S(M_S)$ from $R_i$ and act as follows:*
  - *If $\mathcal{S}_S(M_S)$ is verified, then $S$ is faulty, as it signed the bad $\mathcal{MAC}_{k_s}(M||n_s)$ and*

*should be banned from the network.*

- *If $\mathcal{S}_S(M_S)$ is not verified, then $R_i$ did not properly verify the signature before sending the message (or forged the message itself), hence $R_i$ should be banned from the network.*

*Unlike the first simple protocol, in which the server uses no PKI operations, the second extended protocol requires PKI operations only when misbehaving node behavior is detected, and D wishes to identify the dishonest node. Hence, even the second extended protocol with PKI requires almost no effort on the destination server end. Additionally, signing and verification of signatures made by the source client and relay require both to perform some expensive operations, and can also be used to provide some proof-of-work. The actual banning from the network can be performed by sending signed revocation lists by D to the various overlay relays. Finally, the destination server can do bookkeeping, and record what the different nodes in the overlay did. In case financial incentive was used, the destination server can act as the trusted party and deliver payments or credit according to the actual relaying that took place.*

### 3.3.4 Overlay Design Goals

*Our design strives to meet three main design goals. First we would like to be able deliver packets even over congested links, utilizing the real delivery probability $P_D$; see Eq. 1.1.*

*Second, we must refrain from self-creating DoS ourselves while trying to utilize $P_D$. Obliviously we cannot transmit data without any congestion control mechanism, and we would always prefer using TCP over UDP.*

*Third, we wish to harness additional overlay resources, which we can utilize to increase delivered capacity and reduce packet delivery delay. We expect that in some cases we would relay through nodes with higher $P_D$, thus expediting packet delivery. Otherwise, we settle for the additional bandwidth.*

### 3.3.5 Overlay Design

*All nodes discussed in this chapter are assumed to be a part of a managed overlay which cooperates to deliver data to the destination. Unlike traditional relaying overlays, which are used to simply relay the data to the destination, possibly using multipath, such as proposed by Stavrou et al. [72], the key idea of our overlay is aggregating bandwidth, to break through to the destination, as well as persisting in packet delivery, especially when facing a bandwidth-flooding DDoS. Unlike persistence in QoSoDoS [33], using an overlay may assist in routing via less congested routers, such that the same transmission rate may be more effective. We acknowledge that if our scheme is misused it may help in DDoS amplification, hence we assume that our overlay is carefully controlled, especially in regards to the total transmission bandwidth of the overlay, as described in Section 3.3.6. Furthermore, by controlled overlay we mean that in case the destination is purposely unreachable, the attack is too large or otherwise the overlay behavior is unwanted, it can be taken off by the overlay controller.*

*To avoid redundant transmissions, we assume that a fountain-code (FC) such as RaptorQ [53] can be used. FCs are forward error correction (FEC) codes, that can generate from the source message as many encoding symbols as needed. The receiver should be able to decode the source message in very high probability from almost any set of encoding symbols of sufficient cardinality – which is about the number of source symbols or very slightly more. Hence, in our overlay, sources and relays send different encoding symbols to the destination. Once enough packets are received, the receiver reconstructs the original message, and the message transmission stops. Acknowledgements from the victim can be sent using UDP, as the congestion is on the victim's downlink, whereas its uplink is completely under its control.*

*FCs are prone to become a source for DoS, as the destination keeps state in order to reconstruct the original message. Consequently, an attacker may try to exhaust the server's memory (state) and/or processing capabilities (reconstructing). To address this, we assume that packets' sources are authenticated, allowing the server to differentiate between*

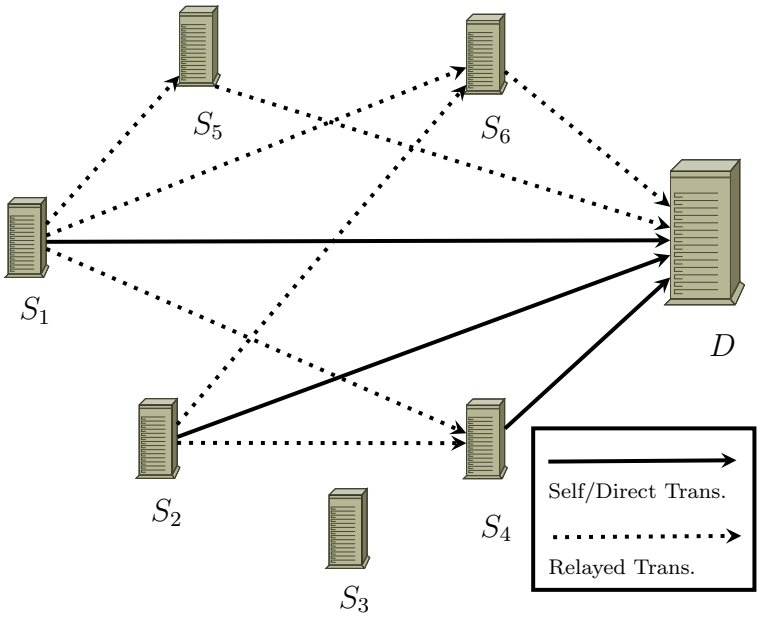



Figure 3.2: Overlay design. $S_1..S_6$ are overlay nodes and $D$ is the destination. Nodes with self-generated data use their bandwidth to transmit solely their own data (solid lines), e.g., $S_1 \rightarrow D$, $S_2 \rightarrow D$, and $S_4 \rightarrow D$. Otherwise, nodes may act as relays (dashed lines), e.g. $S_5$ and $S_6$. Relaying nodes may relay data from more than a single source, in which case their route will be (fairly) shared, e.g. $S_6 \rightarrow D$ is shared by both $S_1$ and $S_2$.

*legitimate clients and attackers, and ban misbehaving clients, as described in Sections 3.3.1-3.3.3. Additionally, FCs should not be used for messages containing only a few packets, such as DNS and short HTTP requests/responses, as simply retransmitting the packet avoids the overhead introduced by FCs, i.e., the extra packets and (re)construction. Although short messages may encounter (significant) redundancy, thanks to their small bandwidth requirements, they should still be cost-effective.*

*In each transmission the source node optimistically begins to transmit using TCP, in hopes that TCP's throughput will be at least some minimal rate, denoted* $R_{min}^{TCP}$*. Once TCP rate drops below* $R_{min}^{TCP}$ *the node is allowed to begin transmitting using UDP. The UDP rate is limited to a predefined rate, denoted* $R_{max}^{UDP}$*. Note that the TCP connection is not aborted, however the UDP transmission stops once TCP's rate is back above* $R_{min}^{TCP}$*. The expected goodput is approximately TCP's rate plus UDP's goodput, i.e., approximately* $R_{max}^{UDP} \cdot P_D$*.*

*When using UDP, a node may request help from overlay nodes in relaying its data.*

*Relaying is done like regular transmissions, i.e. using TCP and optionally UDP if the TCP rate drops below $R_{min}^{TCP}$. However, like P2P systems, we assume that overlay nodes are self-interested, and if they have self-data to deliver they will deliver it using their entire bandwidth, without sharing it with other nodes. If a node is not transmitting its own data, it may use its bandwidth for relaying. The overlay is fully distributed, and no centralized server is controlling it, as to avoid attacks on the centralized controller. Hence, more than one node may ask the same relay to deliver its data. We assume that the relay bandwidth is fairly shared among all requesting nodes, however, this cannot be assured.*

*Relaying through one or more overlay nodes might detour the congested links, and possibly be able to relay the information using TCP. Such a TCP relaying network is called an Indirect Overlay Network (ION), e.g. RON [9], OverQoS [74] and Bandwidth-Aware Routing in Overlay Networks [50]. Note that traditional IONs assume good TCP performance, and merely bypass packet losses, congestion, or bandwidth limitations. However, if all routes to the destination are congested, IONs are likely fail in data relaying. Our solution can utilize congested routes, even such with low $P_D$ and bad performance.*

### 3.3.6 Building an Amplification Overlay

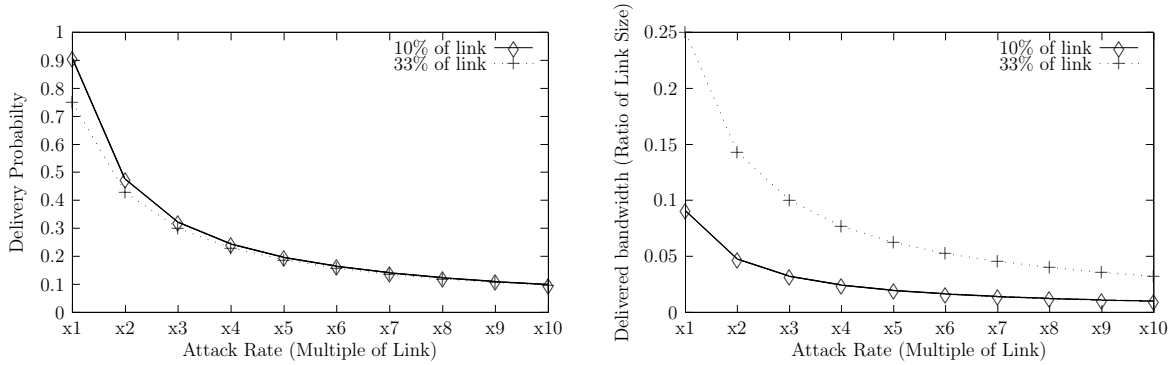
*It is important to keep in mind that the total UDP bandwidth of the overlay must be carefully controlled, otherwise, if the overlay network's permitted bandwidth grows too large it may cause congestion collapse, and prolong the congestion indefinitely. Hence the overlay total allowed bandwidth must not reach a critical point in which it can amplify an attack once the attacker stopped its attack or reduced its rate.*

*Generally speaking, if we would like to cause an attacker to utilize, or cause utilization [48, 70] of more bandwidth than the overlay, i.e., reduce the attack amplification factor, we should make sure that the overlay bandwidth does not exceed 50% of the (congested) link's rate. Hence, assuming we use UDP rate which is twice the minimal TCP rate, i.e. $R_{max}^{UDP} = 2 \cdot R_{min}^{TCP}$, and stop UDP transmissions once TCP reaches its minimal rate, we*

*can say that the total rate of the non-congestion controlled UDP transmissions, made by the entire overlay, should not exceed* $1/3$ *of the bottleneck link. The reason is that* $1/3$ *of UDP rate +* $1/6$ *of TCP rate, totals to* $1/2$ *of the link's rate. This way to sustain an attack, the attacker must produce more than 50% of the bottleneck link's rate. Moreover, since we do not expect that the entire overlay will transmit at the same time, this would probably won't be enough on behalf of the attacker.*

*Due to the Internet's topology and decentralization, the bottleneck link is almost always the "last-mile" link(s) connected the destination, hence it should be easy to identify and limit the overlay's bandwidth accordingly.*

*Inherent in the overlay size and link capacity utilization is a trade-off between the goodput/throughput and the transmission rate. The higher the transmission rate is, the less the delivery probability, as* $R_I = R_A + R_C$*, see Eq. 1.1, where* $R_A$ *and* $R_C$ *are the attacker's and clients' rate respectively. This implies that the more bandwidth we use to deliver the packets, the less relative data (goodput/throughput) will go through. Figure 3.3 depicts this tradeoff. The figure depicts the theoretical delivery probability (Fig. 3.3(a)) and theoretical goodput (Fig. 3.3(b)) when using 10% and 33% of the link's rate.*




(a) Delivery Probability ($P_D = \frac{R_O}{R_I}$, $R_I = R_A + R_C$)

(b) Client's Goodput ($R_C \cdot P_D$)

Figure 3.3: Theoretical overlay throughput vs. delivery probability (3.3(a)) and goodput (3.3(b)) tradeoff. The x-axis is the attacker's bandwidth as a multiple of bottleneck link. Larger overlay throughput contributes to the delivery probability reduction, thus degrading the goodput to throughput ratio. The figure shows that an overlay transmitting at 33% of the link's bandwidth causes up to 25% loss of its throughput, whereas an overlay delivering 10% of the link's bandwidth loses less than 10% of its throughput.

## 3.4 Evaluation

*We try to evaluate two main things. First, we would like to estimate the severity of the BW-DDoS problem and the vulnerability of ASes to the sheer volume of bandwidth attacks reported to date. Second, we would like to estimate the effectiveness and trade-offs of our proposed solution.*

### 3.4.1 Simulator Design and Implementation

*We have developed a simulator which uses CAIDA's AS relationships dataset [3] to construct an Internet-scale AS topology. CAIDA's data is derived from RouteViews [76] BGP table snapshots, and includes the following inferred AS relationships: customer-provider, peer-to-peer, and sibling-to-sibling. AS relations capture the inter-AS agreements which dictate which routes can be traversed and which cannot. Currently, our simulator uses the* shortest traversable paths.

*We are not aware of any available AS-to-AS link bandwidth data, hence we have adopted the philosophy proposed by Songjie et al. [81], in which large ASes will have bigger links than small ASes, and that AS size is proportional to its connections degree. Using the obtained topology, the simulator uses the links as described in Table 3.1. We consider three types of node sizes, small (up to 4 links), medium (between 5 and 300 links) and large (more than 300 links).*

*Traffic simulation itself is divided into two parts, based on the observations made in Figure 1.1. Bandwidth is initially used by UDP transmissions, followed by TCP transmissions which use only the remainder of bandwidth. Thus, if 100% of the bandwidth is used by UDP, we assume no TCP transmissions take place.*

*In each simulation step a set of flows is used, such that each flow tries to deliver some amount of data over a specific route. The data may contain UDP traffic, TCP traffic, or both, depending on the configuration. Flows can be dynamically added or removed prior to each simulation step. Each route consists of a list of links, and each link has a knowledge*

*of its outgoing bandwidth and flows going through it – that is, all its incoming traffic. The simulation simulates data flows as a stream of bits, which should produce an approximate mean value for packet transmission.*

*To capture that UDP is prioritized over TCP, as described in Section 3.2, simulation steps are composed of two parts: computing the UDP traffic going through all links, followed by computing TCP traffic. UDP traffic is calculated by iterating the flows, and calculating the amount of data delivered by each link along the route. In case a link is congested, that is, it has a higher incoming bandwidth than outgoing bandwidth, then the amount of UDP bandwidth passed to the next link along the flow's route is reduced proportionally to the delivery probability $P_D$ (Eq. 1.1). Therefore, the succeeding links receive less bits and are updated accordingly. This process is iterated along the route, until reaching the destination. This implies that UDP traffic is the same or reduced over the route, i.e. preceding links in the route carry the same or more UDP traffic than succeeding links. If instead incoming traffic is larger than outgoing traffic and there's available bandwidth, then each flow's traffic is proportionally increased, up to a maximum of its incoming traffic. The process is iterated, such that in each iteration there's recalculation of $P_D$ for each link compared to the previously computed $P_D$. Once the change in delivery probability is less than some predefined $\epsilon$, we assume the UDP calculation has stabilized and stop the iterating. In our simulations we set to $\epsilon = 0.01$*

*After UDP traffic has stabilized, TCP traffic is calculated by iterating all flows containing TCP traffic, and trying to send all the TCP traffic. Each link calculates its unused outgoing bandwidth, that is the outgoing bandwidth after reducing the UDP transmitted over that link. It then splits the bandwidth proportionally between incoming TCP traffic. In case the amount of TCP traffic is changed, the* entire *flow's TCP rate is updated accordingly. This implies that all links of a specific TCP flow deliver the same TCP bandwidth. This differs from the non-increasing/decreasing rate of UDP flows. Similar to UDP, the process is iterated until changes are smaller than $\epsilon$.*

*We used the simulator to test two scenarios. The first is AS availability as a function*

| Src/Dst | **Small** | **Medium** | **Large** |
|---|---|---|---|
| **Small** | OC-3 | OC-12 | OC-24 |
| | 155.52 Mbps | 622.08 Mbps | 1.244 Gbps |
| **Medium** | OC-12 | OC-48 | OC-192 |
| | 622.08 Mbps | 2.488 Gbps | 9.953 Gbps |
| **Large** | OC - 24 | OC - 192 | OC - 768 |
| | 1.244 Gbps | 9.953 Gbps | 39.813 Gbps |

Table 3.1: Bandwidth estimation based on AS size. AS size is estimated based on the number of links it has to other ASes. *Small* AS has up to 4 links. *Medium* AS has up to 300 links. *Large* AS has more than 300 links.

*of attack size. The second is the performance of our overlay scheme.*

### 3.4.2 Availability Simulation

*We have simulated AS connectivity during bandwidth attacks. We recorded a thousand executions of the simulator, and each execution consisted of various sized bandwidth attacks. For each execution we randomly chose a destination AS node, and* $102,400$ *uniformly distributed AS nodes, from which we construct flows for attackers, and an additional* 100 *uniformly distributed nodes to simulate an overlay. Each attacker flow sends* $1Mbps$ *of UDP traffic to the destination, and each overlay node tries to send* $1Mbps$ *of TCP traffic. To simulate various-sized bandwidth attacks, each simulation was executed 10 times, such that each time we double the amount of attacking routes, starting with* 200 *attacking flows, which transmit* $200Mbps$*, up to* $102,400$ *flows which transmit* $102.4Gbps$*.*

*Figure 1.2 presents the results of the simulation. The black bars represent simulator executions in which all overlay nodes were able to transmit some data, using TCP, to the destination. The gray bars are simulations in which some overlay nodes are able to transmit TCP data, while other nodes cannot, implying that some routes are congested and some are open. In these situations, nodes with congested flows can try to detour the congested links, by finding an overlay node with a non-(fully) congested route, and relay*

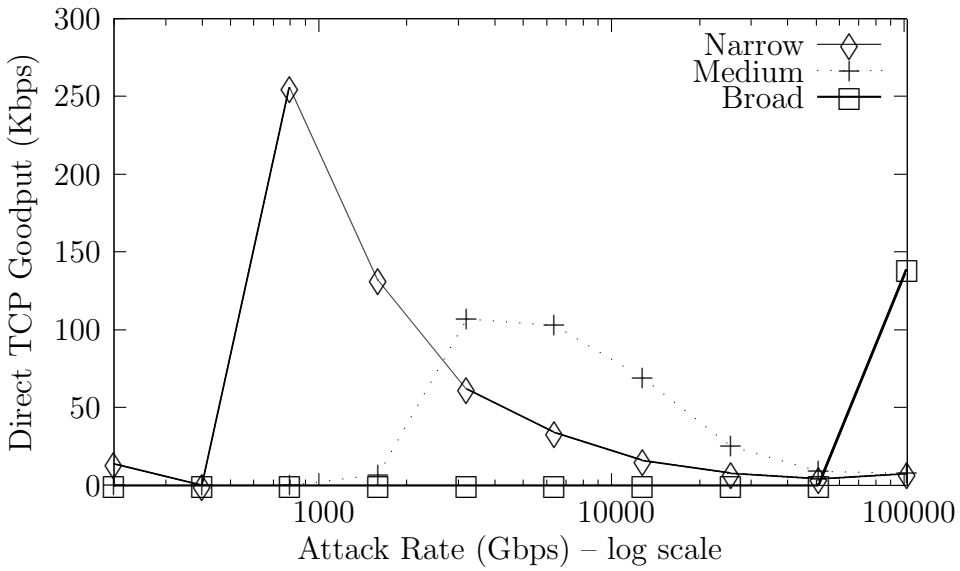



Figure 3.4: AS clusters based on total bandwidth as per Table 3.1. *Narrow bandwidth* consists of ASes with total bandwidth up to 1.6 *Gbps*. *Medium bandwidth* consists of ASes with total bandwidth between 1.6 *Gbps* and 12.8 *Gbps*. *Broad bandwidth* ASes consist of ASes with total bandwidth over 12.8 *Gbps*.

*data through the uncongested node to the destination, i.e., use a routing overlay. The white bars are the simulations in which all routes to the destination are congested, and no route can sustain TCP flows to the destination, which implies that no detouring route exists, while our solution can enable communication.*

### 3.4.3 Overlay Performance Simulation

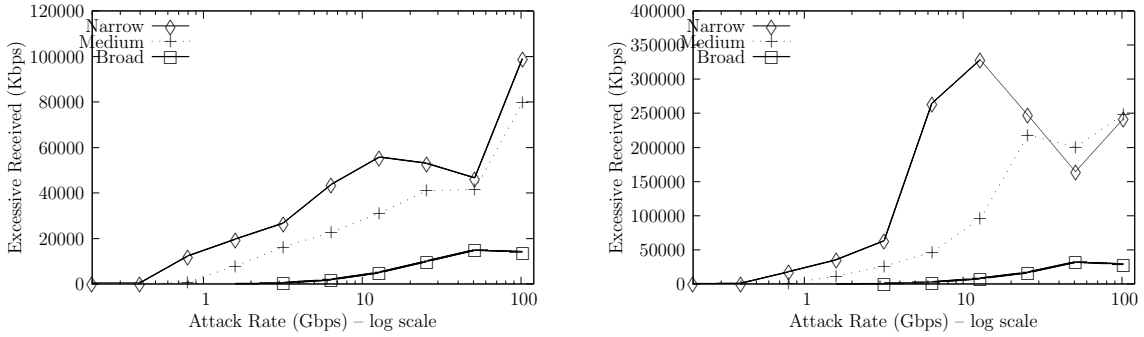
*For this simulation we have implemented the overlay as described in Section 3.3. Due to the diversity of the Internet topology and link sizes, the simulation results exhibit large variance. Consequently, we classified the different ASes based on their total links' bandwidth and the largest bandwidth attack they can handle, as depicted in Figure 1.2 (white bars in the figure). The classifications is depicted in Figure 3.4, and shows three types of AS with total bandwidth sizes as follows.* Narrow bandwidth *ASes have less than* 1.6 *Gbps of total bandwidth.* Medium bandwidth *ASes have total bandwidth between* 1.6 *Gbps and* 12.8 *Gbps.* Broad bandwidth *ASes have more than* 12.8 *Gbps total bandwidth.*

*To simulate randomly joining and leaving clients, we configured the simulation such*

*that in each step there's a* 10% *probability that any node in the overlay will have a new message to deliver. In case the node is already sending a message, the new message is queued for deferred delivery. Each message is* 1 *megabit in size. TCP rate is limited up to* $1Mbps$, $R_{min}^{TCP} = 500kbps$ *and* $R_{max}^{UDP} = 1Mbps$. *The overlay consists of 100 nodes, i.e. the total bandwidth producible by the overlay is limited to* $150Mbps$. *Zombies send UDP traffic at* $1Mbps$. *Attack phase of the experiment is* $1,000$ *steps. Sending* 1 *megabit can be typical for file transfer or sending an email with an attachment. For other applications such as web browsing, in which typical HTTP header sizes of 700-800 bytes are common [65], the entire transmission should complete much faster.*

*We present the results of two heuristics used for transmitting a message from source nodes to overlay relay nodes. The first heuristic is called* linear*, that is, it transmits the message to one additional overlay node every step. The second heuristic is called* exponential*, that is, in each step it doubles the number of overlay nodes to which it sends the data.*

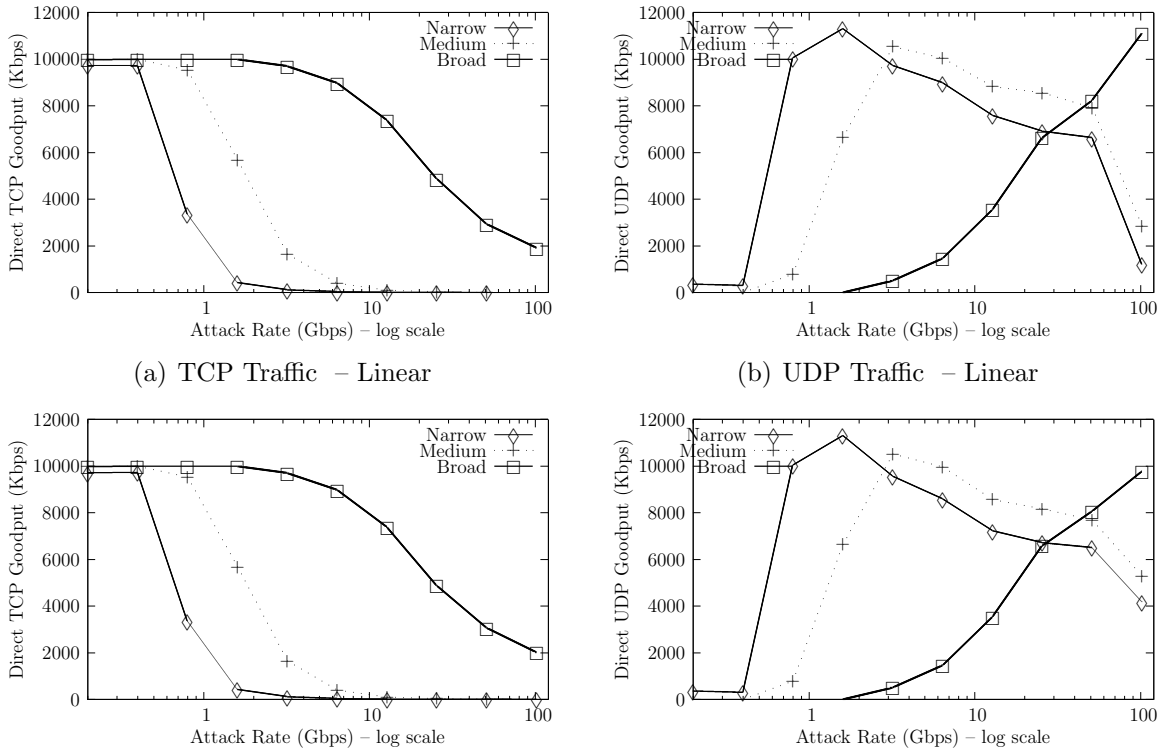
*Both* linear *and* exponential *heuristics exhibit similar behavior. While the time for delivering data using the exponential heuristic is better, compared to the linear heuristic, the exponential heuristic requires significantly more intra-overlay overhead, that is the amount of bandwidth used between overlay nodes and not sent directly to the destination; see results in Figure 3.5.*




(a) Linear (b) Exponential

Figure 3.5: Intra-overlay overhead of data passed between overlay nodes.

*Figure 3.6 depicts the goodput of traffic sent directly from client to server. TCP*




(a) TCP Traffic – Linear

(b) UDP Traffic – Linear

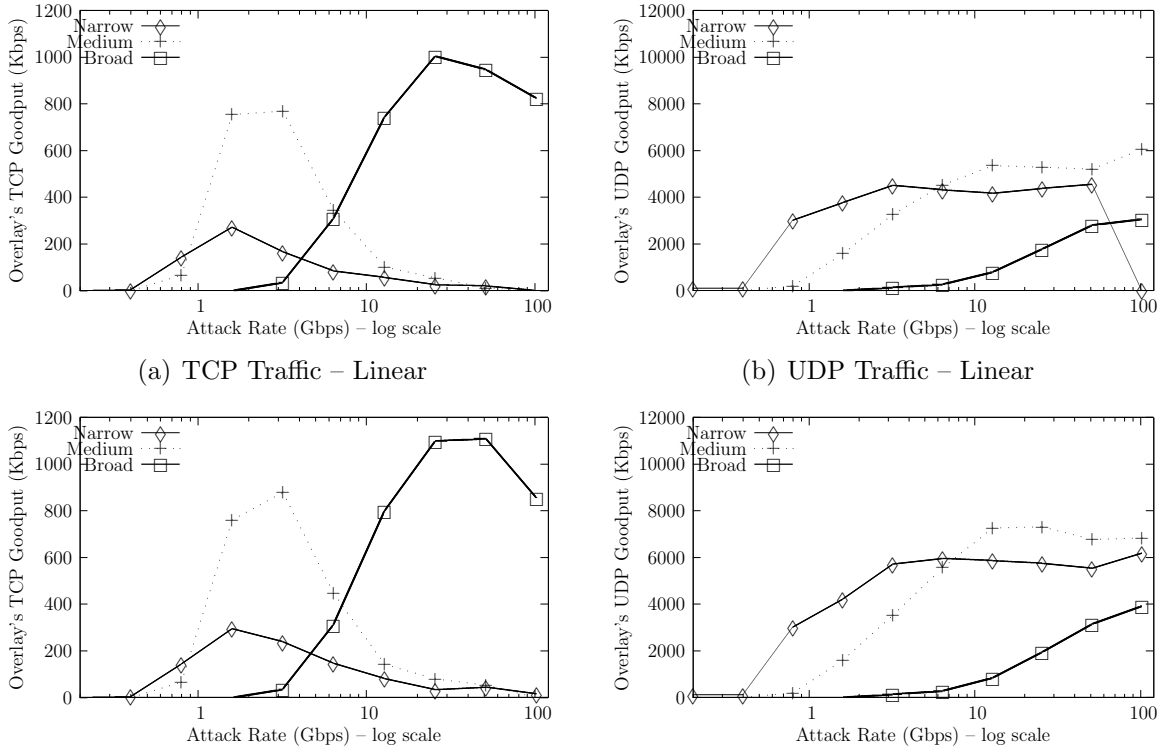
(c) TCP Traffic – Exponential

(d) UDP Traffic – Exponential

Figure 3.6: Goodput traffic sent directly from client to server.

*reduces its rate as a function of attack, and in correlation direct UDP rate increases. For both narrow and medium bandwidth ASes, bandwidth reaches a peak, and from that point onward is reduced, since the delivery probability $P_D$ decreases as a function of attack size. For the broad bandwidth nodes, we observe that they only start using UDP under large attacks, and did not reach their peak within the* $100 Gbps$ *attack. Throughout the results, UDP exhibits in order of magnitude more goodput than TCP, since it utilizes the actual delivery probability, whereas TCP utilizes only the available bandwidth.*

*Figure 3.7 shows that the overlay goodput of TCP reaches the peak for narrow, medium, and broad bandwidths ASes, whilst the exponential heuristics manages to use slightly more TCP traffic than the linear. However, using UDP, large-scale attacks manage to reduce the effectiveness, at least for narrow bandwidth, whereas the exponential is still able to keep growing. The reason for this can be explained using Figure 3.8, in which we can observe that for the very large-scale attacks, the majority of the nodes in the overlay*




(a) TCP Traffic – Linear

(b) UDP Traffic – Linear

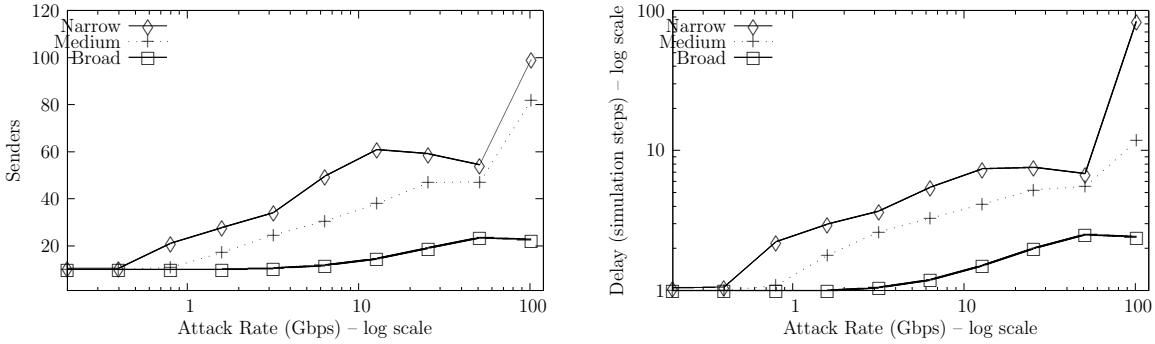
(c) TCP Traffic – Exponential

(d) UDP Traffic – Exponential

Figure 3.7: Overlay goodput traffic sent via a relay from client to server.

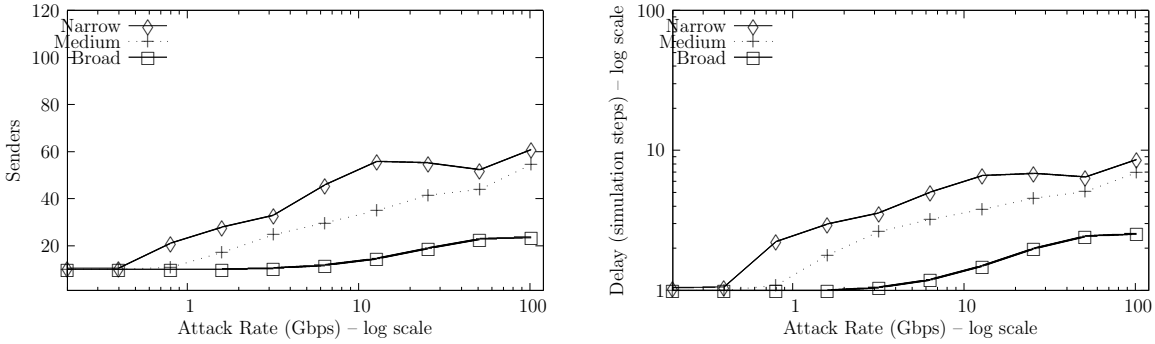
*are transmitting their own data. That is, there are almost 100 senders, but none of them would relay any data, rendering the overlay ineffective. This observation is especially true when considering the intra-overlay traffic, see Figure 3.5, which approaches* 100 *Mbps at that point. On the other hand, since the average exponential heuristic transmission is shorter, there are less senders on average; see Figure 3.8. We assume that for larger attacks than those we have simulated, a similar effect will happen for the exponential heuristic as well. This observation sheds light on the trade-off between the probability of sending a node's own data, and the time to deliver a message. The smaller the probability that an overlay node would not share its bandwidth, or alternatively as more nodes are available for delivering data, the more effective the overlay would be. Note that this effect seems soon to happen in larger provisioned ASes with medium bandwidth.*

*Similarly, Figure 3.9 depicts the amount of completed transmissions in each simulation step. The degradation correlates to the reduction in the effectiveness of using the overlay.*




(a) Senders – Linear

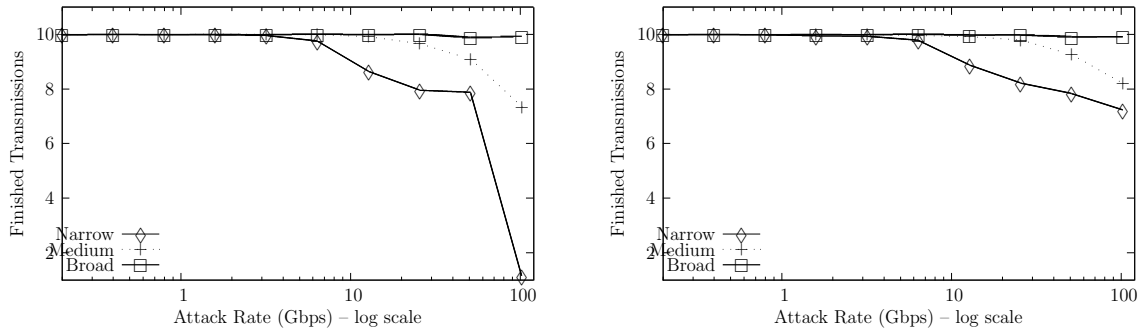
(b) Delay (simulation steps) – Linear




(c) Senders – Exponential

(d) Delay (simulation steps) – Exponential

Figure 3.8: Senders vs. delivery latency per transmission (in simulation steps).



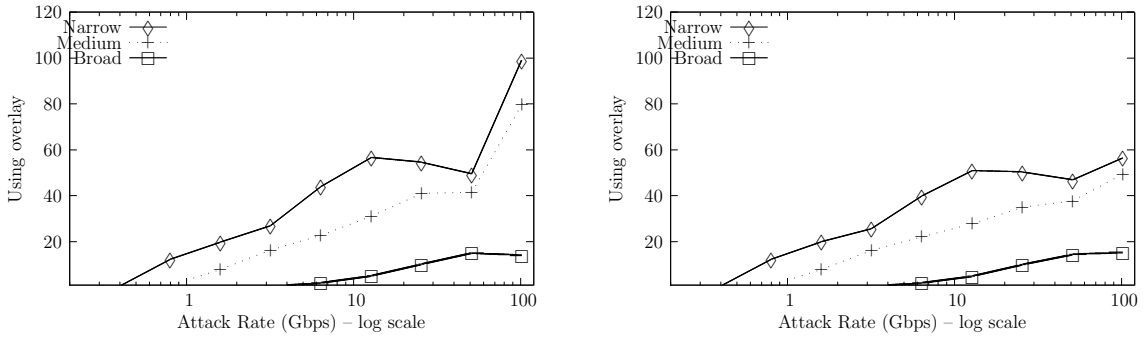

(a) Linear

(b) Exponential

Figure 3.9: Average Completed Transmissions

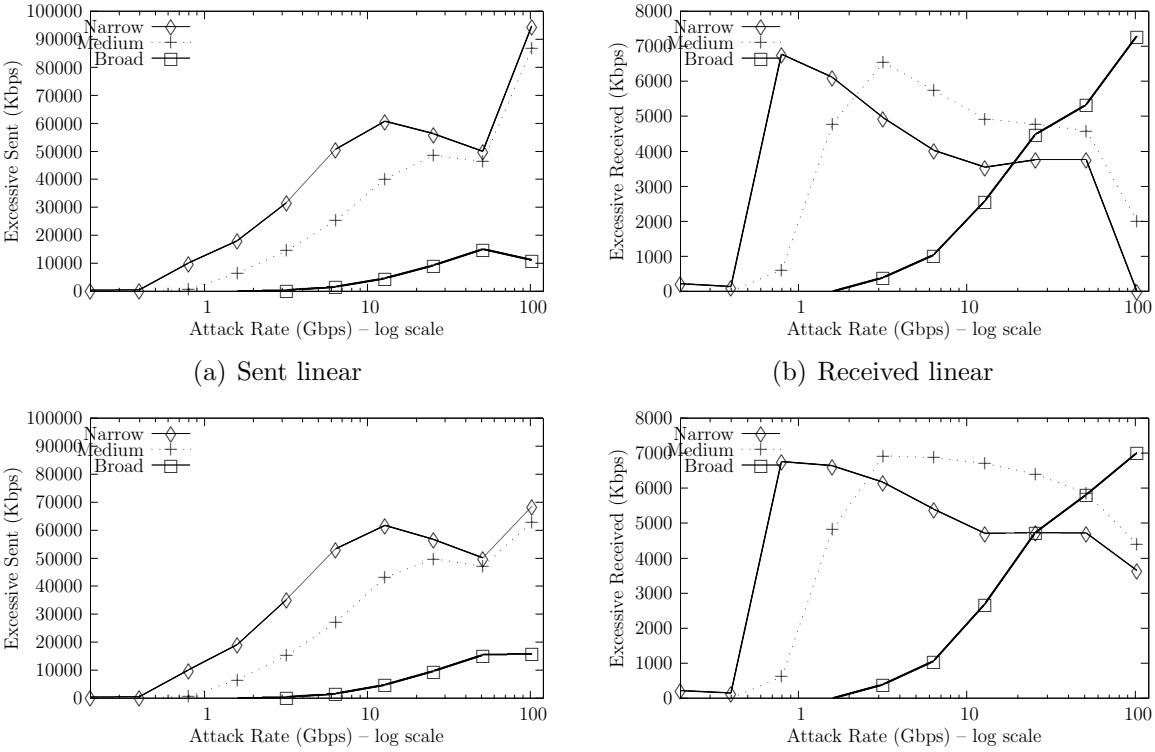
*Once all nodes are trying to transmit data using the overlay, we only get the delivery probability of the sender vs. the attack size.*

*Figure 3.11 depicts the amount of excessive data being sent to the destination, and the amount of excessive data which actually reaches the destination, which is in orders of magnitude less, due to the low delivery probability* $P_D$.




(a) Linear (b) Exponential

Figure 3.10: Senders trying to use the overlay.




(a) Sent linear (b) Received linear

(c) Sent exponential (d) Received exponential

Figure 3.11: Excessive sent and received traffic (overhead)

## 3.5 Conclusions

*In this chapter we have presented a novel mechanism which uses overlays to mitigate flooding attacks. Previous work, including overlay solutions, have concentrated on filtering the offending flows, absorbing the attack in various ways, or circumventing the congested links. These solutions are themselves prone to large-scale bandwidth attacks, made by an*

*attacker with bandwidth larger than they can handle. Our work presents a complementary solution which lets a controlled overlay cooperatively send information, without congestion control, thereby competing against the attackers' packets, and substantially increasing legitimate packet delivery.*

*We presented design and Internet-scale simulation results, based on real world AS topology. The simulation results show that our solution can mitigate flooding DDoS attacks even at the scale of the largest attacks seen to date.*

# Chapter 4

# Backward Traffic Throttling

*We present* Backward Traffic Throttling *(BTT), an efficient, decentralized mechanism for mitigation of congestion and bandwidth-flooding attacks. Upon congestion, BTT employs two basic mechanisms to throttle excessive traffic, namely: prioritize and shape legitimate flows traffic, and pushback, or request upstream BTT routers to similarly prioritize and shape traffic. Flow prioritizing parameters are independently determined by each BTT server, based on typical traffic estimations. BTT requires deployment in routers, but is relatively easily deployed: it requires no changes to routers' hardware or software, and does not modify the packet traffic. Instead, BTT configures routers based on existing mechanisms in the router; specifically, BTT configures queuing disciplines and traffic shapers.*

*Both simulation and testbed experiments were performed to estimate the effectiveness of BTT during BW-DDoS attacks. Results reported in Section 4.3 show that even limited BTT deployment alleviates attack damage and allows legitimate TCP traffic to sustain communication, whereas larger deployments maintain larger portions of the original bandwidth.*

## 4.1 Introduction

*Distributed denial-of-service (DDoS) attacks, and in particular bandwidth-flooding DDoS attacks, are hard to prevent and mitigate in a decentralized, best-effort network such as the Internet. Mitigation is hard, also due to the huge amount of resources exploited by the attacker during DDoS attacks, launched by large botnets. Recent studies showed dramatic growth of DDoS attack volumes [12], e.g., during 2010, volumes exceeded 100 Gbps.*

*Bandwidth flooding botnets usually send traffic to victim hosts, causing packet losses on bottleneck links, thereby making TCP and TCP-friendly flows drop their transmission rates. New attack types utilizing peer-to-peer (P2P) traffic between compromised nodes, such as Coremelt [73], pose an additional threat. Depending on the botnet size, P2P attacks can utilize small traffic rate between many pairs of zombies (bots), and cause congestion on a link between two routers (possibly at the core of the network); consequently such Coremelt attacks can be hard to detect and block.*

*Router defenses rely on various router mechanisms. Prominent operational router-based tools include Access Control Lists (ACL), Remote-Triggered Black-Holes (RTBH), firewalls and FlowSpec [12]. Other proposed solutions include* capabilities *and* rate-limiting *schemes. Capabilities schemes, such as TVA [85] and SIFF [84], propose a DoS-limiting network architecture in which destination and routers use secure token (capability) information, indicating the destination's willingness to receive information from the source. Rate-limiting schemes, such as PSP [22], limit flows rate during attack back to their typical rate, thereby reducing congestion. For comparison between the various defense schemes see Table 1.2.*

*This chapter details the design, simulations and testbed experimentation of* Backward Traffic Throttling (BTT)*, a novel* distributed *mechanism, deployed within autonomous system (AS) networks. BTT design is based on carefully combining two mechanisms:* throttling *of traffic causing congestion at routers, and a cooperative* pushback *protocol, which requests upstream routers to perform corresponding throttling, closer to the traffic*

*sources. Throttling further consists of two mechanisms, namely changing routers' queuing discipline and traffic shaping. BTT throttles network traffic based on observed congestion, or when requested by an authenticated downstream BTT server. Request based throttling is done only for traffic which will be routed via the AS who issued the throttling request. Throttling is adjusted to preserve both* fairness *and* link utilization*. may reduce their rates considerably upon loss detection. In the BTT context, fairness is provisioning sources with their typical transmission rates, which are estimated prior to the attack. When the excessive traffic which is causing congestion subsides, BTT reduces and then stops throttling.*

*Rather than modifying the internals of routers, BTT is deployed using servers which do not handle the traffic flow; the servers configure routers queuing discipline and traffic shapers as needed. Hence, unlike previous solutions (see Table 1.2), BTT does not require any changes to routers, or any manipulations to packets. BTT is decentralized, i.e., it can be adopted independently by one or (preferably) more Autonomous Systems (AS), incrementally improving defenses against DDoS attacks.*

*BTT makes the following design choices:*

**Fairness** *BTT uses estimates of the typical traffic to divide the available bandwidth among the incoming links, achieving fair allocation, as per the typical rate.*

**Decentralization** *BTT uses decentralized traffic prioritization, i.e., downstream BTT servers only request upstream BTT shapers to comply with rate limitations, but the upstream BTT server can decide which packets to discard if any, and which further upstream shaping to announce. In addition, the typical rate estimation is also carried out in a decentralized manner.*

**Flow obliviousness** *BTT throttles traffic based on weights derived from the typical traffic rates, avoiding the need to identify the actual attacking flows, making BTT robust and not attack-specific. For example, BTT is resilient to spoofing attacks since it does not consider the flows' sources based on their source IP address. Instead it simply limits atypical excessive traffic, based on its actual flooding source. Similarly, BTT does*

*not differentiate between the Slashdot effect (flash crowd), and DDoS; BTT always assures fairness as per the typical rate. Any unused bandwidth will be fairly shared among demanding flows.*

Evaluation. *We performed both simulations based on realistic AS topology, as well as experiments in the DETERlab testbed [16]. Both simulation and testbed experiments results show that BTT significantly prioritizes legitimate traffic over attack traffic, even in massively distributed scenarios. The results show that even in limited deployments, BTT proves to be effective. Furthermore, as BTT adoption increases, an increase in the amount of legitimate traffic is also observed, demonstrating that even partial and gradual deployment of BTT proves valuable, and provides an effective defense against flooding DDoS.*

*The rest of the chapter is organized as follows. Section 4.2 presents design of BTT. In Section 4.3 we present an empirical evaluation of BTT. Conclusions are presented in Section 4.4.*

## 4.2 Design

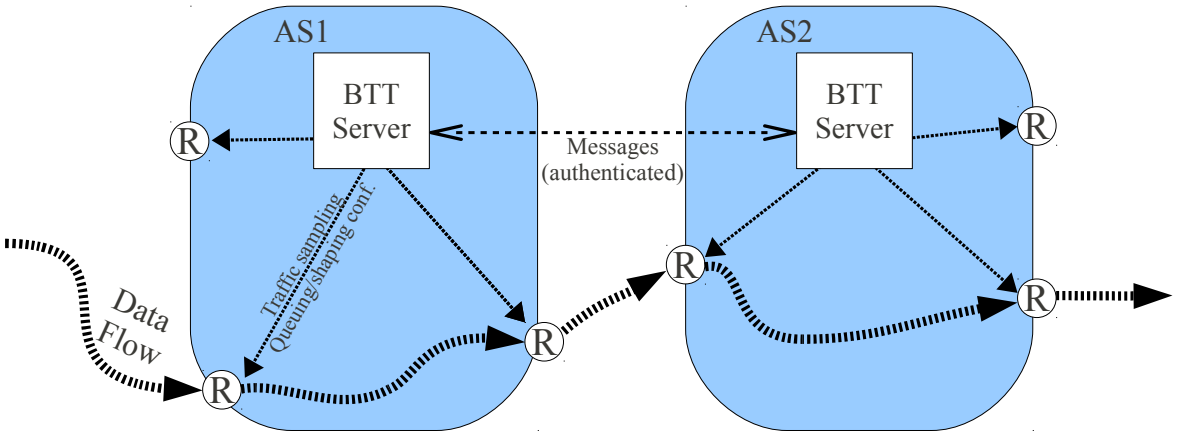



Figure 4.1: BTT system architecture.

*Figure 4.1 depicts the system architecture. BTT servers continuously read traffic statistics from the AS routers, which are used to estimate the typical traffic prior to*

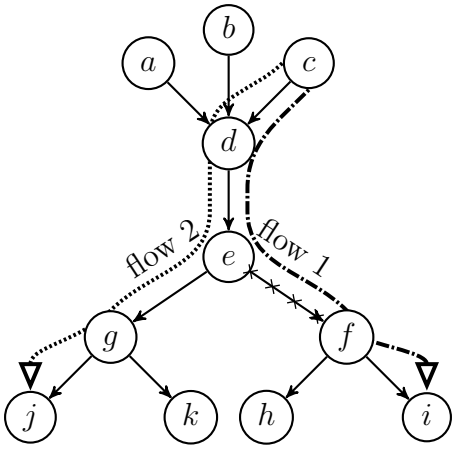



Figure 4.2: Backward throttling example. Operation steps: (1) Node $e$ detects congestion on link $\langle e, f \rangle$. (2) Node $e$ activates throttling on incoming links. (3) Node $e$ notifies node $d$. (4) Node $d$ creates a virtual link of the traffic destined to $\langle e, f \rangle$ and throttles it. The traffic of flow 2 will not be throttled as it is not destined to $\langle e, f \rangle$. Note that the throttling of a flow only depends on its downstream route.

*an attack and identify congestion. Traffic estimations should be stored in an AS-wide database, which can contain a significant amount of traffic statistics and can be queried during an attack.*

*BTT servers control the routers' queuing and shaping policies, which by default are left unchanged. Whenever BTT detects congestion on some link, or if it receives a shaping request from a peer BTT, it prioritizes and shapes the traffic traversing the congested link. Traffic prioritizing and shaping is configured on the routers themselves using standard queuing disciplines and shapers. Next, BTT propagates rate limiting notifications to upstream BTTs, which should perform corresponding traffic shaping and limit the attack closer to its sources.*

### 4.2.1 Network Model

*The network is modeled as a directed graph, with Autonomous Systems (ASes) as nodes. We assume that each node knows the AS-level route that each outgoing packet will traverse and that no routing changes occur while the throttling is active. The routes can be learned, for example, from the* AS_Path *sequence of the Border Gateway Protocol (BGP) [67].*

*We denote a link from AS a to AS b as an ordered pair* $L = \langle a, b \rangle$. *The set* $UP(L)$ *contains all neighboring links upstream of* $L$, *which contribute to the traffic from* $a$ *to* $b$. *Each AS can measure the current traffic rate,* $CurRate^{L_D}(L)$, *received on an incoming link* $L$ *and destined to traverse a downstream link* $L_D$.[1] *Note that* $L_D$ *might be several hops away from* $L$.

### 4.2.2 Typical Traffic and Weights

*The basic idea behind BTT is to utilize* typical traffic rates *estimation as a guideline for fairly splitting the available bandwidth among upstream neighbors. Similarly to other papers which proposed using typical traffic estimation, we assume that it is indeed possible to collect "typical traffic" statistics. In this chapter we rely on existing methods, e.g., as proposed in PSP [22].*

*BTT keeps typical traffic estimations, denoted* $Typ^{L_D}(L_U)$, *per upstream link* $L_U$ *routed through downstream link* $L_D$. *These estimations should be calculated and stored for different times of the day, and updated periodically, e.g. daily, possibly using techniques such as exponential smoothing and giving lower weights to recent data, as to prevent an attacker from easily messing with the statistics.*

*To estimate the maximum storage volume of the typical traffic tables, we are required to estimate the number of ASes in the Internet. According to the CAIDA [3] dataset, there are* $36,878$ *ASes, hence in the worst case we need to keep information about* $O(AS^2)$ *source-destination AS pairs, i.e.,* $O(36,878^2)$. *Assuming we store hourly average/maximum leaky bucket parameters per day of the week per source-destination AS pair, each record contains 100 bytes, which is more than required to keep the leaky bucket parameters, we get approximately 20TB we need to store. These are worst case limits as no AS should be in route of all Internet traffic, hence no AS should keep track of the whole 20TB, let alone that 20TB is a reasonable volume for most, if not all, ASes to store.*

[1] For readability, time is omitted from our notation.

### 4.2.3 Congestion Handling

*When identifying congestion in a downstream link $L_D$, BTT fairly allocates a fraction $\phi_{L_U}^{L_D}$ of its $L_D$'s rate for each upstream link $L_U$ which delivers traffic via $L_D$. The weights are derived from the typical traffic estimation, as described in Eq. 4.1.*

$$\phi_{L_U}^{L_D} = \frac{Typ^{L_D}(L_U)}{\sum_{L \in UP(L_D)} Typ^{L_D}(L)} \tag{4.1}$$

*In the current design of BTT, we use wighted fair queuing (WFQ) to allocate the typical rate. In WFQ, if any flow does not fully utilize its bandwidth share, then the remaining bandwidth will be (fairly) utilized by the other flows. This implies that as long as there is any demand, the available bandwidth will be properly utilized.*

### 4.2.4 Backward Traffic Throttling

*Each AS executes an independent instance of BTT. Each BTT instance begins traffic throttling when an outgoing link is congested, i.e. exceeds some rate threshold, or when receiving a* Notify *message from a downstream AS.*

*If BTT was initiated due to congestion, then it starts using WFQ followed by sending* Notify *messages to neighboring BTT servers. If the throttling was activated due to a* Notify *message, then BTT creates a virtual link, $L_D$, containing all traffic routed through the congested link; see Figure 4.3. Both virtual and physical links are treated similarly. For a physical links, the rate capacity of $L_D$, denoted $C^{L_D}$ is $L_D$'s bandwidth. For a virtual links $C^{L_D}$ is the rate restriction requested by the* Notify *message. If the*

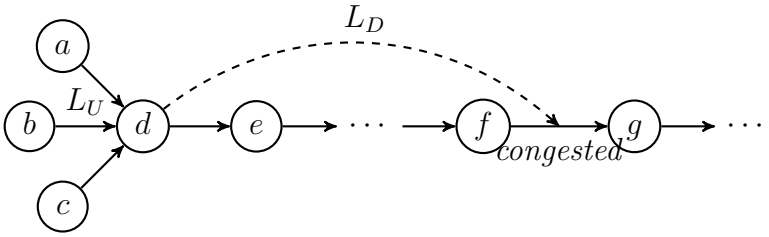



Figure 4.3: Schematic view of topology. $L_D$ is a virtual link containing all traffic from $d$ to the congested link $\langle f, g \rangle$. $L_U$ is a neighbor upstream link of $d$.

*current rate exceeds $C^{L_D}$, BTT uses WFQ for incoming traffic, using weights as defined in Equation 4.1.*

*If as a result of the upstream shaping the congested link becomes underutilized, BTT slowly increases the shaping limits, until $L_D$ has high utilization. The increase in shaping is based on the available underutilized capacity of $L_D$. Links receive additional bandwidth quotas as a function of their weight $\phi_{L_U}^{L_D}$.*

*To describe the shaping increase, we need to calculate the harshest restrictions for incoming links. For simplicity, we start by describing the basic allocation scheme, which limits each incoming link to a fraction of $C^{L_D}$. To that end we describe the typical* overcapacity factor *of $L_D$, denoted $OCF^{L_D}$, where:*

$$OCF^{L_D} = \frac{C^{L_D}}{\sum_{L \in UP(L_D)} Typ^{L_D}(L)} \tag{4.2}$$

*For example, if $L_D$ is a 100 Mbps link, with total typical traffic of 50 Mbps, the overcapacity factor will be $OCF = \frac{100}{50} = 2$. In the worst case, an incoming link $L_U$ is restricted to $\phi_{L_U}^{L_D} \cdot C^{L_D}$, which corresponds to its typical rate multiplied by the overcapacity factor of $L_D$. For example, an incoming link with typical rate of 10 Mbps, might be restricted to as low as $\frac{100}{50} \cdot 10 = 2 \cdot 10 = 20$ Mbps.*

*Using the harshest shaping restrictions may lead to underutilization of $L_D$, when some incoming links transmit less than their shaping restrictions. Therefore, an adaptive* overbooking factor*, denoted $OBF^{L_D}$, is maintained for each downstream link $L_D$, adjusting the restrictions over time. The bandwidth restriction for $L_D$, denoted $Rest^{L_D}(L_U)$, is defined as:*

$$Rest^{L_D}(L_U) = \phi_{L_U}^{L_D} \cdot OBF^{L_D} \cdot C^{L_D} \tag{4.3}$$

*Initially, $OBF^{L_D}$ is set to 1, hence $Rest^{L_D}(L_U) = \phi_{L_U}^{L_D} \cdot C^{L_D}$, and each upstream link $L_U$ receives its fair share of $L_D$ rate. Next, we periodically check whether $OBF^{L_D}$ needs to be adjusted, increasing or decreasing the shaping restrictions, as described in Equation 4.6. To that end we derive a rate usage parameter, $\rho^{L_D}$.*

$$\rho^{L_D} = \frac{C^{L_D}}{\sum_{L_U \in UP(L_D)} \min\{CurRate^{L_D}(L_U),\ Rest^{L_D}(L_U)\}} \tag{4.4}$$

*A BTT-compliant upstream link would deliver a maximum rate of* $Rest^{L_D}(L_U)$*, hence* $CurRate^{L_D}(L_U) \leq Rest^{L_D}(L_U)$*. For non-compliant links* $CurRate^{L_D}(L_U)$ *may be larger than* $Rest^{L_D}(L_U)$*. To prevent non-compliant links from overusing bandwidth at the expense of compliant links, we regard non-compliant links as if they were only using their full restriction, that is, a minimum between* $CurRate$ *and* $Rest$*. If the bandwidth is not fully utilized, we let compliant links increase their rate.*

*Using Equation 4.5, BTT derives* $\bar{\rho}^{L_D}$*, the change factor of* $OBF^{L_D}$*.* $\bar{\rho}^{L_D}$*'s "aggressiveness" is restrained by using two configurable parameters:* $\alpha$*, and a maximum change,* $\bar{\rho}_{max}$*. In our experiments we have set both* $\alpha$ *and* $\bar{\rho}_{max}$ *to* 2*.*

$$\bar{\rho}^{L_D} = \begin{cases} \min\left\{\bar{\rho}_{max},\ 1 + (\rho^{L_D} - 1)/\alpha\right\}, & \text{if } \rho^{L_D} \geq 1 \\ 1 + (\rho^{L_D} - 1) \cdot \alpha, & \text{if } \rho^{L_D} < 1 \end{cases} \tag{4.5}$$

$$OBF^{L_D} \leftarrow \max\left\{1,\ \bar{\rho}^{L_D} \cdot OBF^{L_D}\right\} \tag{4.6}$$

*Examining Equations 4.4 and 4.5, we see that when* $\rho^{L_D} \geq 1$ *then* $C^{L_D}$ *is larger than the used capacity, hence* $L_D$ *is underutilized, and we will increase the permitted rate for upstream BTTs, in an inverse proportion to* $\alpha$*. Otherwise, if* $\rho^{L_D} < 1$*, then* $L_D$ *is overutilized and BTT reinstates restrictions in direct proportion to* $\alpha$*. Note that to have any affect on OBF,* $\rho^{L_D}$ *must be larger than* $1 - \frac{1}{\alpha}$*, otherwise* $\bar{\rho}^{L_D} < 0$*. Therefore, Based on Eq. 4.6, we will set* $OBF^{L_D} = 1$*, which forms our initial restriction per link* $L_U$ *of* $\phi_{L_U}^{L_D} \cdot C^{L_D}$*. This implies that a legitimate client transmission restriction is at least overcapacity times more than their typical rate, i.e.:*

$$Rest^{L_D}(L_U) \geq OCF^{L_D} \cdot Typ^{L_D}(L_U) \tag{4.7}$$

*BTT would gradually increase utilization of underutilized links by "moving" the unused bandwidth capacity to demanding upstream links, gradually allowing them to send more and more traffic. Note that even while loosening the restrictions, BTT does not change the WFQ weights. Thus, attacker rate spikes or oscillations should not significantly affect BTT-compliant upstream links.*

*Updating* $OBF^{L_D}$ *changes the restrictions, as per Equation 4.3, and should lead to messages sent between BTT servers. Messages which decrease the senders' rates are sent immediately, whereas rate increase messages are sent periodically. Hence, BTT quickly responds to congestion, but delays relaxation of active restrictions.*

### 4.2.5 Attack Recovery

*If BTT sees that the congestion dropped below some threshold, it restores the original queuing discipline, followed by sending* NOTIFY *with infinity rate restriction. To avoid exploitation by the attacker which changes its attack rate from high to low at high frequencies, two practices are used. First, restrictions are removed slowly as described above. Second, BTT activation is made at a high kick-in threshold* $\Theta$ *and the deactivation process begins only if the link utilization drops below a lower threshold,* $\theta$*. Both* $\Theta$ *and* $\theta$ *are configurable, whereas* $\theta$ *should be between the typical link utilization and* $\Theta$*. In any case, even if BTT stops shaping, whenever the rate re-crosses* $\Theta$*, BTT automatically re-initiates the throttling, limiting such an attack impact.*

## 4.3 Evaluation

*We performed both testbed experiments and simulations. The experiments were designed to test how BTT handles real traffic and to measure its performance. The simulations were designed to test BTT on an Internet-scale topology, and its behavior when only partially deployed.*

| Type | Param | Description | Section |
|---|---|---|---|
| **Links** | $L_U$ | Upstream (originating) link | 4.2.2(Fig.4.3) |
| | $L_D$ | Downstream (congested) link | 4.2.2(Fig.4.3) |
| | $UP(L)$ | Set of neighboring links upstream of $L$ | 4.2.1 |
| | $C^{L_D}$ | Capacity (bandwidth) of link $L_D$ | 4.2.4 |
| **Rate** | $Typ^{L_D}(L)$ | Typical traffic from $L$ flowing through $L_D$ | 4.2.2(Fig.4.3) |
| | $CurRate^{L_D}(L)$ | Current rate from $L$ flowing through $L_D$ | 4.2.1 |
| | $Rest^{L_D}(L)$ | Current restriction on $L$ for traffic flowing through $L_D$ | 4.2.4 |
| **Factors** | $\phi_{L_U}^{L_D}$ | Typical fraction of $L_U$ traffic in $L_D$ | 4.2.2(Eq.4.1) |
| | $OCF^{L_D}$ | Overcapacity factor | 4.2.4(Eq.4.2) |
| | $OBF^{L_D}$ | Overbooking factor | 4.2.4(Eq.4.6) |
| | $\rho^{L_D}$ | Rate usage factor | 4.2.4(Eq.4.4) |
| | $\bar{\rho}^{L_D}$ | $OBF^{L_D}$ change factor | 4.2.4(Eq.4.5) |

Table 4.1: BTT Notation Summary Table.

### 4.3.1 Emulation Setup

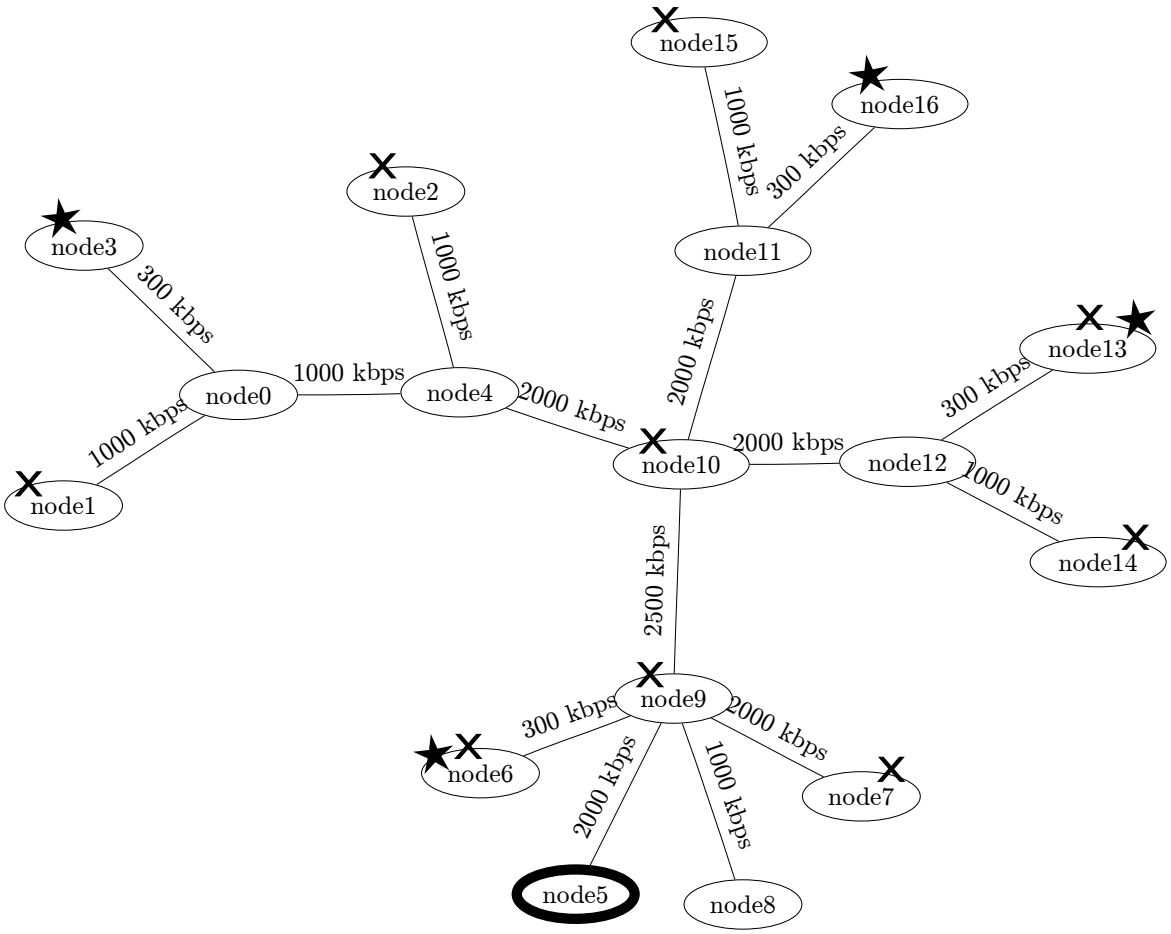



Figure 4.4: Testbed topology. A star marks legitimate TCP sources. An *X* marks a UDP attacker. Traffic destination is *node*5 for both legitimate and attack traffic. Weights are links' bandwidth. During experiments, *node*16 rate was upgraded to 600kbps.

*To evaluate the performance of BTT in a real environment, we developed a prototype in Python that emulates a router with all the features of BTT (BTT server, traffic estimator, traffic scheduler, and traffic shaper). The prototype was tested on the DETERlab testbed [16], using the topology shown in Figure 4.4.*

*The attack traffic was generated using DETERlab's traffic generation tools, using constant-bit-rate of* 1*KB-packets over UDP. The nodes from which attack traffic is generated are marked with an* X *in figure 4.4. All the traffic, both attack and legitimate, is directed to node*5*.*

*The legitimate traffic is produced using FTP connections uploading a large file to*

*node*5. *Each node marked with a star maintains 3 FTP connections. Prior to the experiment, we measured the rates of the TCP connections to be used as the typical traffic rates. In order to test how BTT handles legitimate traffic changes, the TCP bandwidth of node*16 *was doubled to* 600*kbps* after *the typical rate estimates were calculated.*

*Each testbed node used Ubuntu 10.04 (Linux 2.6.32), with the standard TCP/IP protocol stack. The nodes were connected via links with drop-tail queues of length* 20 *(packets). Links bandwidths are rate limited, as depicted in Figure 4.4.*

### 4.3.2 Testbed Results

*Figures 4.5–4.10 depict traffic measurements between node*16 *and node*5. *Figures 4.5 and 4.6 show the behavior without activating BTT. Figure 4.5 presents the aggregated rate of the 3 FTP connections from node*16*, before, during and after the attack. Figure 4.6 depicts the delay and packet loss as measured using* `ping`. *Both figures constitute a baseline for the following results.*

*Figures 4.7 and 4.8 depict experimental results in which BTT is enabled, but overbooking is disabled. Figures 4.9 and 4.10 depict experimental results in which both BTT and overbooking are enabled. As depicted in Figures 4.7 and 4.8, the activation of BTT succeeds in maintaining a good throughput of the legitimate streams, even without the use of overbooking. When using overbooking, BTT was able to restore TCP to its original rate, even during the attack, as depicted in Figures 4.9 and 4.10.*

*Figure 4.11 depicts the bottleneck link utilization, when BTT is enabled with overbooking, while being able to maintain the typical traffic of legitimate clients (Figures 4.9, 4.10)*

*Note that the presented rate originates from a source node which is not generating attack traffic. Nodes that do generate attack traffic, e.g., node*13*, cannot maintain their TCP connections rate using the current implementation of BTT without introducing a filtering mechanism, or some kind of an intra-node fair-queuing.*

### 4.3.3 Simulation Setup

*In addition to testbed experiments, we have adapted the PAWS simulator [80], and used it to evaluate BTT. PAWS uses a full-scale AS-level Internet topology, including links, bandwidths and routing information, as described by Wei et al. [82].*

*For each setup of legitimate sources, attackers and BTT nodes, the simulator iterates for several rounds, until it reaches a stable network state. In each round, packet delivery ratios are calculated for all the links in the network, based on the rates transferred in the previous round. Using these ratios, the traffic of the current round is adequately adapted.*

*Legitimate traffic is simulated as TCP-friendly flows, altogether consuming the minimum between the bottleneck capacity and the flow's maximal rate. For each simulation execution, approximately* $350,000$ *flows are created, by randomly choosing pairs of ASes and setting their maximal rate to* $1\%$ *of the available bandwidth along the path between them. This method results in moderate link utilization.*

*Attack traffic is generated by a botnet, constructed based on the Slammer worm [62] distribution, using* $50,000$ *bots, such that the bandwidth of all the bots is configured to create a certain overload to the target link.*

*Our simulation setup was made of the following:*

**Attack overload factor** *The attacks are designed to congest a single target link, while possibly causing collateral congestion at peripheral links. The* attack rate *is the botnet's rate arriving at the incoming queues of the target link. The* attack overload factor *is the* attack rate *divided by the* target link's bandwidth. *An* attack overload factor *above* $1$ *causes packet loss on the link. In the simulation we used an attack factor of* $2$ *which simulates packet delivery probability of about* $50\%$*, implying that any TCP connection would have very low throughput, and possibly even disconnect.*

**BTT deployment ratio** *BTT is deployed at a random set of ASes.* BTT deployment ratio *is the ratio of the deployed ASes out of the total number of ASes.*

**Goodput ratio** *To measure the degradation of legitimate traffic rates during an attack, we calculate the ratio between the legitimate rate* during *the attack and the legitimate*

*rate* before *the attack. This* goodput ratio *is calculated for the target link, such that a low ratio indicates that the attack is successful. Since legitimate flows are TCP-friendly, a ratio below* 100% *but above* 0% *means that the attack consumes only a portion of the link's bandwidth, letting legitimate traffic utilize the remainder of the bandwidth. That given, an attack which overloads the link above its bandwidth results in a goodput ratio close to* 0%.

### 4.3.4 Simulation Results – Attacks on Stub Links

*A classic attack type targets a single AS (e.g., an AS containing a Web server), usually connected by a single link to a provider AS. To test this scenario, we choose random target links for which the tail degree is* 1 *and the head degree is below* 10. *These links represent a connection between a small-sized AS and a stub AS. In the simulated attack all the bots send traffic to the stub AS, causing packet loss on the target link.*

*Figure 4.12 depicts BTT performance enhancement as its deployment widens. BTT provides significant mitigation when deployed on all ASes. When deployed only at the AS adjacent to the target (first point above* 0 *in the figure) poor performance is observed, as the attacking traffic overloads all the incoming links as well.*

## 4.4 Conclusions

*In this chapter we have presented BTT, a deployable, efficient, decentralized mechanism to mitigate bandwidth-flooding DDoS attacks. BTT is implemented as a server outside the traffic flow. BTT collects typical traffic statistics prior to the attack, and uses it during DDoS attacks to throttle the bandwidth of incoming links. Throttling is based upon on-the-fly configuration of existing queuing disciplines and traffic shapers within routers. BTT does not require changes to any protocol, router's hardware or software, or any packet manipulations.*

*Testbed experimentation and Internet AS scale simulations indicate that BTT can*

*throttle a high percentage of the attack rate and sustain a considerable amount of goodput.*

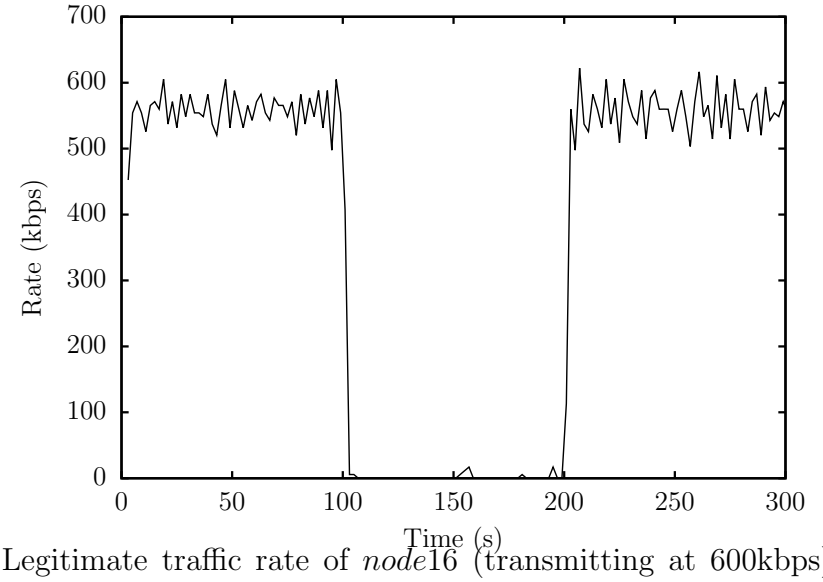



Figure 4.5: Legitimate traffic rate of *node*16 (transmitting at 600kbps) when BTT is disabled. The attack takes place between $t = 100$s and $t = 200$s. TCP traffic drops to practically zero during the attack.

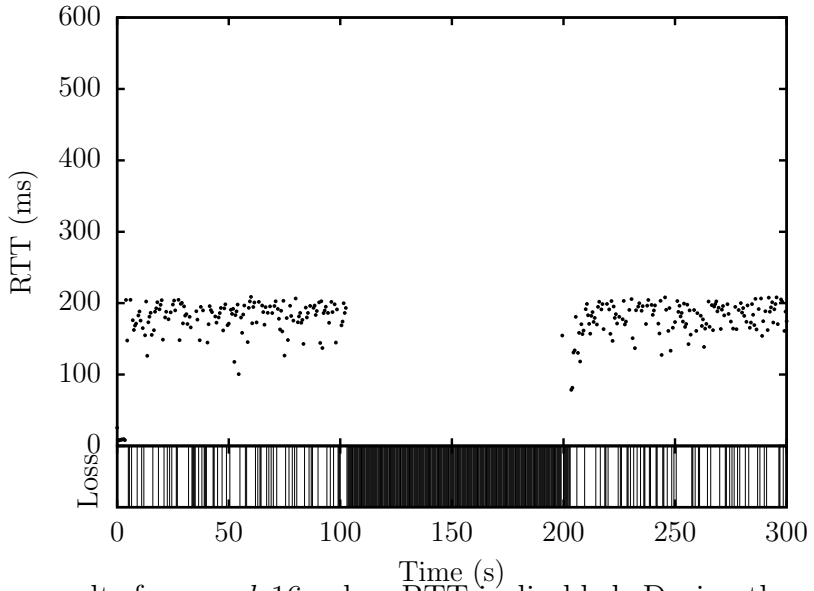



Figure 4.6: Ping results from *node*16, when BTT is disabled. During the attack, all ping packets were discarded. Ping rate is 10 packets per second, 1KB each. When an echo-reply packet is received, the RTT will appear as a dot in the upper section, otherwise a line will be marked in the lower section.

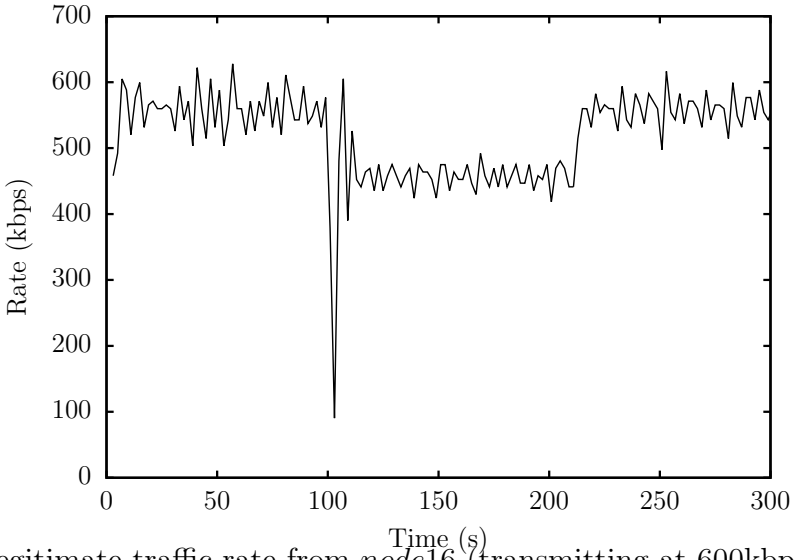



Figure 4.7: Legitimate traffic rate from *node*16 (transmitting at 600kbps), when BTT is enabled, *without overbooking*. Two seconds after the attack begins, BTT restores TCP to about 90% of its rate.

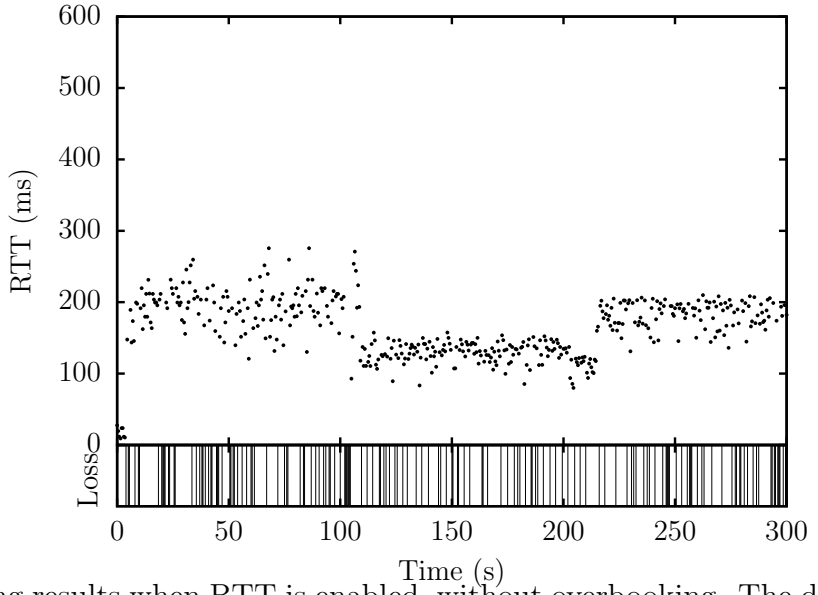



Figure 4.8: Ping results when BTT is enabled, without overbooking. The delay decreases during the attack, as BTT without overbooking restricts the traffic rates under their normal rates.

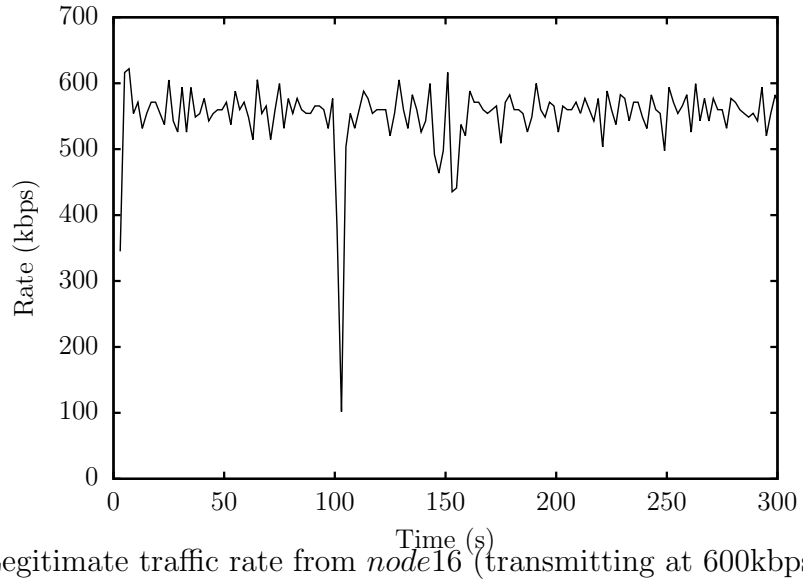



Figure 4.9: Legitimate traffic rate from *node*16 (transmitting at 600kbps), when *BTT is enabled with overbooking*. The legitimate TCP is restored to its normal rate even during the attack.

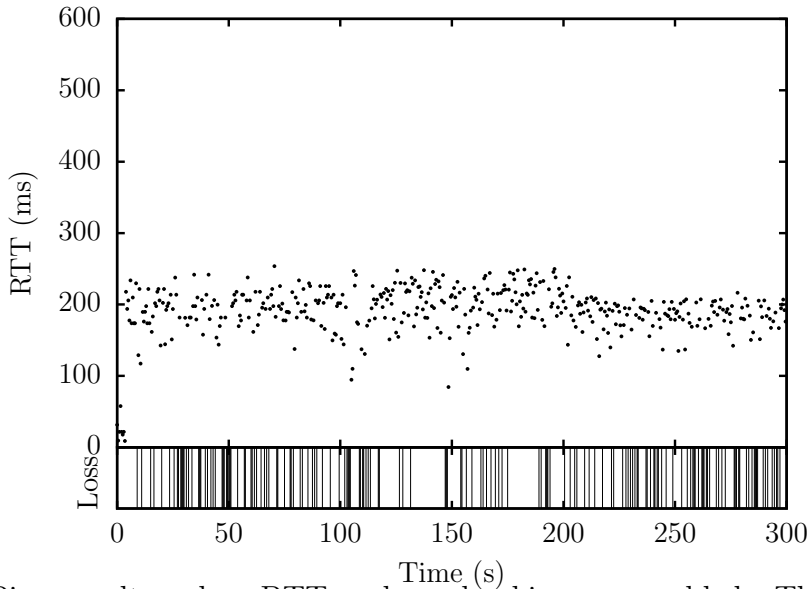



Figure 4.10: Ping results, when BTT and overbooking are enabled. The delay is not affected by the attack.

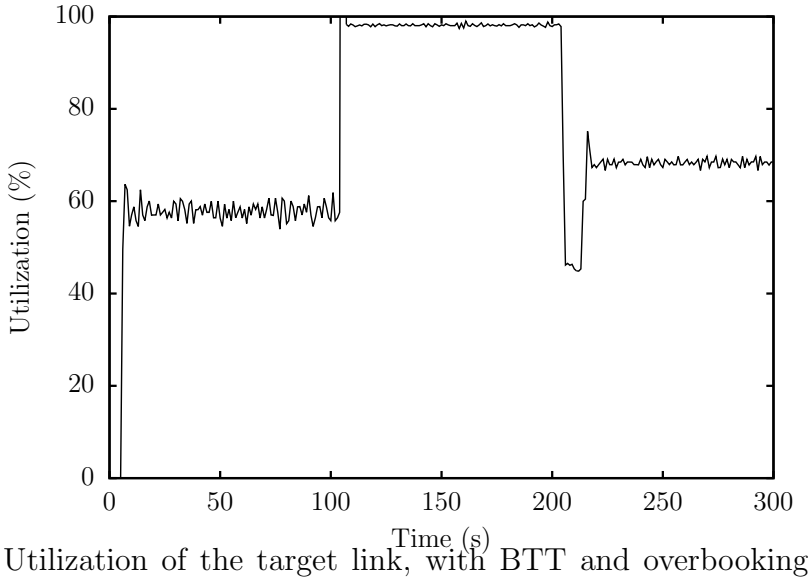



Figure 4.11: Utilization of the target link, with BTT and overbooking enabled in the testbed experiments. Under attack, BTT succeeds in fully utilizing the target link.

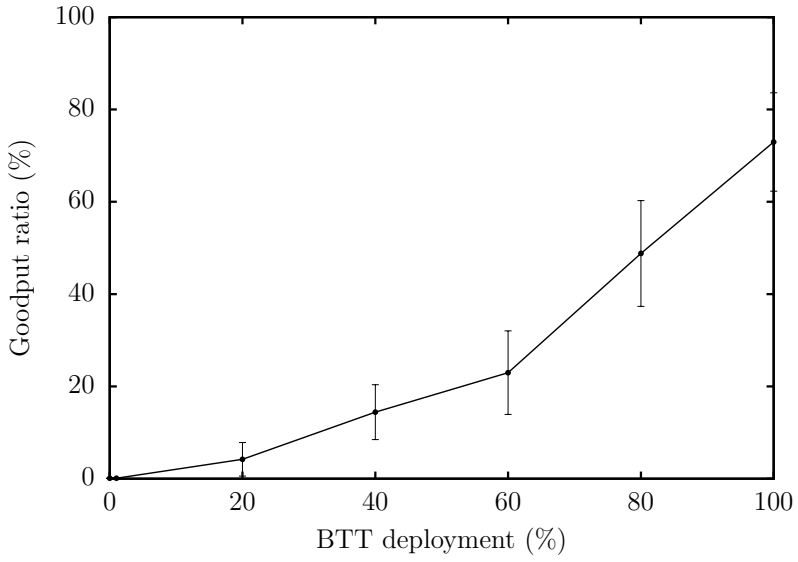



Figure 4.12: The goodput ratio vs. BTT deployment ratio, when the attack is against a stub link, using the Slammer distribution. The legitimate traffic was randomized. The attack overload factor was set to 2.

# Chapter 5

# Conclusions

*BW-DDoS consist of the majority of DDoS attacks [41], which in turn are found at the top of the security concerns of Tier 1, Tier 2, and other IP network operators around the world [12]. In Chapter 1, we have reviewed various BW-DDoS attacks and defenses. So far, BW-DDoS have employed relatively crude, inefficient, "brute force" mechanisms. However, known attacks, which are not commonly used, allow attackers to launch sophisticated attacks, which are difficult to detect and may considerably amplify attackers' strength. Furthermore, the recent largest BW-DDoS attacks have indeed used more advanced techniques; this may indicate that attackers may adopt more effective, advanced BW-DDoS attacks in the future.*

*We argue that deployed and proposed defenses may struggle to meet increasing threats, hence more advanced defenses should be deployed. This may involve some proposed mechanisms (not yet deployed), as well as new approaches. Note that some of the proposed defenses may raise operational and political issues; these are beyond the scope of the current thesis, but should be carefully considered. Finally, we argue that in order to become practical, defense mechanisms should be easy to deploy and would require minor changes, if any, to the Internet's infrastructure routers.*

*In Chapters 2-4, we have presented three schemes which can be deployed in today's Internet for improving QoS, and mitigation of BW-DDoS. QoSoDoS and controlled over-*

*lays are end-host-based and can be deployed without any Internet service providers' intervention, hence, can be implemented by anyone. On the other hand, BTT requires deployment at the infrastructure level as well as cooperation between AS/ISP. Nevertheless, even though the deployment is at the infrastructure level, it can be done externally to the data-plane, and merely requires the ability to configure routers without the need to make any hardware or software changes to equipment which is commonly difficult to upgrade.*

*In Chapter 2 we presented QoSoDoS, a protocol that ensures (modest) Quality of Service, among peers connected via an unreliable, BW-DDoS-prone network, which* usually *has much higher bandwidth, which is the common situation in the Internet. QoSoDoS persists in delivering data, and may send each packet many times, until the packet is received correctly by the destination. This mechanism exploits the available delivery probability which, as established and experimentally validated in this thesis, is larger than 0, even for large-scale attacks. When under extreme congestion, QoSoDoS will temporarily stop using congestion control in order to maintain QoS, and resume using congestion control as soon as possible. Despite the fact that QoSoDoS may seem extreme, it should be able to allow selected critical services to survive significant BW-DDoS attacks.*

*In Chapter 3 we presented a new type of cooperative overlay which aggregates the available bandwidth and possibly multiple paths to break through congested networks. We presented a novel design which, upon congestion, turns to an end-host-based overlay to redirect communication and amplify legitimate traffic, using one or multiple paths to the destination as necessary. The overlay is carefully controlled, thus preventing self-generated DDoS. Previous overlays have concentrated on filtering the offending flows, absorbing the attack in various ways, or detouring the congested links. These solutions are themselves prone to large-scale bandwidth attacks made by an attacker with bandwidth larger than they can handle. Our work presents a complementary solution which lets a controlled overlay cooperatively send information, without congestion control, when needed, thereby competing against the attackers' packets, and substantially increasing legitimate packet*

*delivery.*

*In Chapter 4 we presented Backward Traffic Throttling (BTT), an efficient, decentralized mechanism based on router configuration. BTT prioritizes incoming flows, shapes outgoing traffic, and informs upstream BTT nodes to similarly prioritize and shape traffic. BTT requires no software or hardware changes to routers, does not manipulate protocols and does not modify traffic. Instead, BTT configures routers' queuing discipline and traffic shapers. BTT is a deployable, efficient, decentralized mechanism as it is implemented as a server outside the traffic flow. BTT uses typical traffic statistics during BW-DDoS attacks and throttles the bandwidth of incoming links, based upon on-the-fly configuration of existing queuing disciplines and traffic shapers found in routers.*

# Bibliography


[1] *Iperf Network Testing Tool.* `http://iperf.sourceforge.net/`.

[2] *Open DNS Resolver Project.* `http://openresolverproject.org/`.

[3] *The CAIDA AS Relationships Dataset, 2011. http://www.caida.org/data/active/as-relationships/, 2011.*

[4] *Raz Abramov and Amir Herzberg. TCP Ack storm DoS attacks.* Computers and Security*, 33:12–27, 2013.*

[5] *Advanced Network Architecture Group. ANA Spoofer Project.* `http://spoofer.csail.mit.edu/summary.php`*, 2012.*

[6] *Luis Von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: using hard ai problems for security. In* Proceedings of the 22nd international conference on Theory and applications of cryptographic techniques*, EUROCRYPT'03, pages 294–311, Berlin, Heidelberg, 2003. Springer-Verlag.*

[7] *M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), September 2009.*

[8] *M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581 (Proposed Standard), April 1999. Obsoleted by RFC 5681, updated by RFC 3390.*

[9] *David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient overlay networks. In Greg Ganger, editor,* Proc. of the 18th ACM Sympo-

sium on Operating Systems Principles (SOSP-01), *volume 35, 5 of* ACM SIGOPS Operating Systems Review, *pages 131–145, New York, October 21–24 2001. ACM Press.*

*[10] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Rohit N. Rao. Improving web availability for clients with MONET. In* NSDI. *USENIX, 2005.*

*[11] Spiros Antonatos, Periklis Akritidis, Vinh The Lam, and Kostas G. Anagnostakis. Puppetnets: Misusing Web Browsers as a Distributed Attack Infrastructure.* ACM Transactions on Information and System Security, *12(2):12:1–12:15, December 2008.*

*[12] Arbor Networks. Worldwide infrastructure security reports series (2005-2012).* `http://www.arbornetworks.com/report`.

*[13] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033 (Proposed Standard), March 2005. Updated by RFC 6014.*

*[14] Katerina Argyraki and David R. Cheriton. Active internet traffic filtering: real-time response to denial-of-service attacks. In* Proceedings of the annual conference on USENIX Annual Technical Conference, *ATEC '05, pages 10–10, Berkeley, CA, USA, 2005. USENIX Association.*

*[15] F. Baker and P. Savola. Ingress Filtering for Multihomed Networks. RFC 3704 (Best Current Practice), March 2004.*

*[16] Terry Benzel, Robert Braden, Dongho Kim, Cliford Neuman, Anthony D. Joseph, and Keith Sklower. Experience with deter: A testbed for security research. In* TRIDENTCOM. *IEEE, 2006.*

*[17] T. Bonald, M. Feuillet, and A. Proutiere. Is the "Law of the Jungle" Sustainable for the Internet? In* INFOCOM 2009, IEEE, *pages 28–36. IEEE, 2009.*

[18] *Jean-Yves Le Boudec and Patrick Thiran.* Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume 2050 of* Lecture Notes in Computer Science*. Springer, 2001.*

[19] *L.S. Brakmo and L.L. Peterson. Tcp vegas: End to end congestion avoidance on a global internet.* Selected Areas in Communications, IEEE Journal on*, 13(8):1465–1480, 1995.*

[20] *G. Carl, G. Kesidis, R.R. Brooks, and S. Rai. Denial-of-service attack-detection techniques.* Internet Computing, IEEE*, 10(1):82–89, 2006.*

[21] *Yao Chen and Radu Sion. To cloud or not to cloud?: musings on costs and viability. In* Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 29:1–29:7, New York, NY, USA, 2011. ACM.*

[22] *J. C. Y. Chou, B. Lin, S. Sen, and O. Spatscheck. Proactive surge protection: a defense mechanism for bandwidth-based attacks.* IEEE/ACM Trans. Netw.*, 17(6):1711–1723, 2009.*

[23] *Matthew Prince CloudFlare. The DDoS That Almost Broke the Internet.* `http://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet`.

[24] *D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008.*

[25] *David G. Andersen. Mayday: Distributed filtering for internet services. In* 4th USENIX Symposium on Internet Technologies and Systems USITS*, pages 37–39, 2003.*

[26] *T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176.*

*[27] Colin Dixon, Thomas E. Anderson, and Arvind Krishnamurthy. Phalanx: Withstanding multimillion-node botnets. In Jon Crowcroft and Michael Dahlin, editors,* 5th USENIX Symposium on Networked Systems Design & Implementation, NSDI 2008, April 16-18, 2008, San Francisco, CA, USA, Proceedings*, pages 45–58. USENIX Association, 2008.*

*[28] The Measurement Factory. DNS SURVEY: OPEN RESOLVERS.* `http://dns.measurement-factory.com/surveys/openresolvers.html`.

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

*[30] Forrester Consulting Study, commissioned by VeriSign, Inc. DDoS: A threat you can't afford to ignore. http://www.verisigninc.com/assets/whitepaper-ddos-threat-forrester.pdf, 2009.*

*[31] S. Frankel and S. Krishnan. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. RFC 6071 (Informational), February 2011.*

*[32] Yehoshua Gev, Moti Geva, and Amir Herzberg. Backward traffic throttling to mitigate bandwidth floods. In* Globecom 2012 - Communication and Information System Security Symposium (GC12 CISS)*, Anaheim, CA, USA, December 2012.*

*[33] Moti Geva and Amir Herzberg. QoSoDoS: If You Can't Beat Them, Join Them! In* INFOCOM, 2011 Proceedings IEEE*, pages 1278 –1286, April 2011.*

*[34] Moti Geva and Amir Herzberg. DOT-COM: Decentralized Online Trading and COMmerce. In* 8th International Workshop on Security and Trust Management (STM 2012)*, September 2012.*

*[35] Moti Geva, Amir Herzberg, and Yehoshua Gev. Bandwidth distributed denial of service: Attacks and defenses.* IEEE Security & Privacy*, 99, 2013.*

*[36] Yossi Gilad and Amir Herzberg. Lightweight opportunistic tunneling (LOT). In Michael Backes and Peng Ning, editors,* Computer Security - ESORICS 2009, 14th European Symposium on Research in Computer Security, Saint-Malo, France, September 21-23, 2009. Proceedings*, volume 5789 of* Lecture Notes in Computer Science*, pages 104–119. Springer, 2009.*

*[37] D. Grossman. New Terminology and Clarifications for Diffserv. RFC 3260 (Informational), April 2002.*

*[38] Akamai Technologies Inc. Akamai DDoS defender. http://www.akamai.com/html/solutions/security-/ddos_defense.html, 2011.*

*[39] Corero Network Security Inc. Corero network security reports on top 5 ddos attacks of 2011. http://www.corero.com/en/company-/news_and_events?item_id=4, 2011.*

*[40] Corero Network Security Inc. Corero's DDoS defense system (DDS). http://www.corero.com/en/products_and_services-/dds, 2011.*

*[41] Prolexic Technologies Inc. Prolexic Attack Report, Q3 2011 – Q4 2012. http://www.prolexic.com/attackreports, 2011/2012.*

*[42] Trustwave Holdings Inc. Web hacking incident database (whid). https://www.trustwave.com/wp/whid/, 2009.*

*[43] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In* NDSS*. The Internet Society, 2002.*

*[44] Cheng Jin, Haining Wang, and Kang G. Shin. Hop-count filtering: an effective defense against spoofed ddos traffic. In* ACM Conference on Computer and Communications Security*, pages 30–41, 2003.*

*[45] Angelos D. Keromytis, Vishal Misra, and Dan Rubenstein. SOS: an architecture for mitigating DDoS attacks.* IEEE Journal on Selected Areas in Communications*, 22(1):176–188, 2004.*

*[46] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 4340 (Proposed Standard), March 2006. Updated by RFCs 5595, 5596, 6335.*

*[47] W. Kumari and D. McPherson. Remote Triggered Black Hole Filtering with Unicast Reverse Path Forwarding (uRPF). RFC 5635 (Informational), August 2009.*

*[48] Aleksandar Kuzmanovic and Edward W. Knightly. Low-Rate TCP-Targeted Denial of Service Attacks: the Shrew vs. the Mice and Elephants. In* SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 75–86, New York, NY, USA, 2003. ACM.*

*[49] Aleksandar Kuzmanovic and Edward W. Knightly. Low-rate TCP-targeted denial of service attacks and counter strategies.* IEEE/ACM Trans. Netw*, 14(4):683–696, 2006.*

*[50] Sung-Ju Lee, Sujata Banerjee, Puneet Sharma, Praveen Yalagandula, and Sujoy Basu. Bandwidth-aware routing in overlay networks. In* INFOCOM*, pages 1732–1740. IEEE, 2008.*

*[51] Xin Liu, Xiaowei Yang, and Yanbin Lu. To filter or to authorize: network-layer DoS defense against multimillion-node botnets. In Victor Bahl, David Wetherall, Stefan Savage, and Ion Stoica, editors,* Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Seattle, WA, USA, August 17-22, 2008*, pages 195–206. ACM, 2008.*

*[52] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer. Raptor Forward Error Correction Scheme for Object Delivery. RFC 5053 (Proposed Standard), October 2007.*

[53] *M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder. RaptorQ Forward Error Correction Scheme for Object Delivery. RFC 6330 (Proposed Standard), August 2011.*

[54] *Michael Luby. LT codes. In* Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS-02)*, pages 271–282, Los Alamitos, November 2002. IEEE COMPUTER SOCIETY.*

[55] *Ratul Mahajan, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Controlling high bandwidth aggregates in the network.* SIGCOMM Comput. Commun. Rev.*, 32:62–73, July 2002.*

[56] *Ajay Mahimkar, Jasraj Dange, Vitaly Shmatikov, Harrick M. Vin, and Yin Zhang. dFence: Transparent Network-based Denial of Service Mitigation. In* NSDI*. USENIX, 2007.*

[57] *P. Marques, N. Sheth, R. Raszuk, B. Greene, J. Mauch, and D. McPherson. Dissemination of Flow Specification Rules. RFC 5575 (Proposed Standard), August 2009.*

[58] *Jelena Mirkovic, Sonia Fahmy, Peter Reiher, and Roshan K Thomas. How to test DoS defenses. In* Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*, pages 103–117. IEEE, 2009.*

[59] *Jelena Mirkovic and Peter L. Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms.* SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, 2004.*

[60] *Jelena Mirkovic and Peter L. Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms.* Computer Communication Review*, 34(2):39–53, 2004.*

[61] *Jelena Mirkovic and Peter L. Reiher. D-ward: A source-end defense against flooding denial-of-service attacks.* IEEE Trans. Dependable Sec. Comput.*, 2(3):216–232, 2005.*

[62] *D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer worm.* IEEE Security Privacy, *1(4):33–39, 2003.*

[63] *George C. Oikonomou, Jelena Mirkovic, Peter L. Reiher, and Max Robinson. A framework for a collaborative DDoS defense. In* ACSAC, *pages 33–42. IEEE Computer Society, 2006.*

[64] *J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981.*

[65] *Chromium Projects. SPDY: An experimental protocol for a faster web.* `http://dev.chromium.org/spdy/spdy-whitepaper`.

[66] *B. Raghavan and A.C. Snoeren. Decongestion control. In* Proceedings of the Fifth Workshop on Hot Topics in Networks (HotNets-V), *pages 61–66. Citeseer, 2006.*

[67] *Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), January 2006. Updated by RFC 6286.*

[68] *E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. RFC 3031 (Proposed Standard), January 2001. Updated by RFC 6178.*

[69] *Ha Sangtae, Rhee Injong, and Xu Lisong. CUBIC: a new TCP-friendly high-speed TCP variant.* SIGOPS Oper. Syst. Rev., *42:64–74, July 2008.*

[70] *Rob Sherwood, Bobby Bhattacharjee, and Ryan Braud. Misbehaving TCP Receivers Can Cause Internet-Wide Congestion Collapse. In Catherine Meadows and Paul Syverson, editors,* Proceedings of the 12th ACM Conference on Computer and Communications Security, *pages 383–392, pub-ACM:adr, 2005. ACM Press.*

[71] *Amin Shokrollahi. Raptor codes.* IEEE/ACM Transactions on Networking, *14(SI):2551–2567, June 2006.*

[72] *Angelos Stavrou and Angelos D. Keromytis. Countering DoS Attacks With Stateless Multipath Overlays. In* CCS'05: proceedings of the 12th ACM Conference on Com-

puter and Communications Security: November 7-11, 2005, Alexandria, Virginia, USA, *pages 249–259. ACM Press, 2005.*

*[73] Ahren Studer and Adrian Perrig. The Coremelt Attack. In Michael Backes and Peng Ning, editors,* ESORICS*, volume 5789 of* Lecture Notes in Computer Science*, pages 37–52. Springer, 2009.*

*[74] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy H. Katz. OverQos: an overlay based architecture for enhancing internet Qos. In* Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, 2004.*

*[75] J.K. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, and J. Barros. Network coding meets tcp: Theory and implementation.* Proceedings of the IEEE*, 99(3):490–512, 2011.*

*[76] University of Oregon. Route Views Project.* `http://www.routeviews.org/`*.*

*[77] Vaughn, R. and Evron, G. DNS Amplification Attacks.* `http://www.isotf.org/news/DNS-Amplification-Attacks.pdf`*, 2006.*

*[78] P. Vixie. Extension Mechanisms for DNS (EDNS0). RFC 2671 (Proposed Standard), August 1999.*

*[79] Michael Walfish, Mythili Vutukuru, Hari Balakrishnan, David Karger, and Scott Shenker. DDoS defense by offense.* ACM Transactions on Computer Systems*, 28(1):3:1–3:??, mar 2010.*

*[80] Songjie Wei, C. Ko, J. Mirkovic, and A. Hussain. Tools for worm experimentation on the DETER testbed. In* Proc. 5th TridentCom*, 2009.*

*[81] Songjie Wei, J. Mirkovic, and M. Swany. Distributed worm simulation with a realistic internet model. In* Principles of Advanced and Distributed Simulation, 2005. PADS 2005. Workshop on*, pages 71 – 79, june 2005.*

[82] *Songjie Wei and Jelena Mirkovic. A realistic simulation of Internet-scale events. In* Proc. 1st VALUETOOLS*, 2006.*

[83] *Abraham Yaar, Adrian Perrig, and Dawn Xiaodong Song. SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks. In* Proc. IEEE Security Privacy*, pages 130–146, 2004.*

[84] *Abraham Yaar, Adrian Perrig, and Dawn Xiaodong Song. SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks. In* IEEE Symposium on Security and Privacy*, page 130. IEEE Computer Society, 2004.*

[85] *Xiaowei Yang, David Wetherall, and Thomas E. Anderson. A DoS-limiting network architecture. In Roch Guérin, Ramesh Govindan, and Greg Minshall, editors,* Proceedings of the ACM SIGCOMM 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Philadelphia, Pennsylvania, USA, August 22-26, 2005*, pages 241–252. ACM, 2005.*

[86] *David K. Y. Yau, John C. S. Lui, Feng Liang, and Yeung Yam. Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles.* IEEE/ACM Trans. Netw.*, 13(1):29–42, February 2005.*