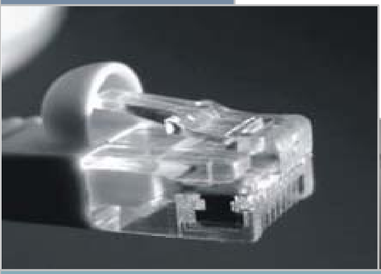
Network Architectures
and Services
NET 2011-06-1

Dissertation

Strategic Resource Management Decisions in Computer Networks

Marc Fouquet

Network Architectures and Services
Department of Computer Science
Technische Universität München



TECHNISCHE UNIVERSITÄT MÜNCHEN
Institut für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

Strategic Resource Management Decisions in Computer Networks

Marc Fouquet

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr. rer. nat. Felix Brandt
Prüfer der Dissertation:	1.	Univ.-Prof. Dr.-Ing. Georg Carle
	2.	Univ.-Prof. Dr. rer. nat. Gabrijela Dreo Rodosek Universität der Bundeswehr München

Die Dissertation wurde am 20.12.2010 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 03.05.2011 angenommen.

Cataloging-in-Publication Data
Marc Fouquet
Strategic Resource Management Decisions in Computer Networks
Dissertation, June 2011
Network Architectures and Services, Department of Computer Science
Technische Universität München

ISBN 3-937201-21-1
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)
Network Architectures and Services NET-2011-06-1
Series Editor: Georg Carle, Technische Universität München, Germany

Abstract

Resource management problems play an important role in technology and economy. In general, resource management tries to use an enterprise's resources as efficiently as possible to maximize a (potentially financial) benefit. This thesis addresses two specific resource management problems in computer networks.

Resource Management during Denial-of-Service Attacks

During a denial-of-service (DoS) attack, an attacker exhausts the resources of one or multiple Internet services by sending large amounts of requests, making the service unavailable. Besides filtering of attack traffic, overprovisioning is a common defense against this attack: The victim tries to have enough resources available to stay operational despite the DoS attack. Traditional resource management techniques like load balancing are not sufficient here, as an attacker does not just create load, but acts strategically to maximize damage to the victim.

Modern computer networks are complex meshes of services that depend on each other. Back end services like DNS or databases are often required to provide the front end services that users are interested in. Such dependencies may create capacity bottlenecks that can be exploited by a DoS attacker to cause more damage with less effort.

In this thesis, we develop methods to uncover such vulnerabilities and to help fix them. To do this, we investigate game-theoretic models of the interaction between attacker and defender. We designed and implemented a network simulator to quickly simulate denial-of-service attacks and use it together with optimization algorithms to find good strategies for attacker and defender. We further show that virtualization of services gives the defender an advantage, as he is able to re-configure his network during an attack. Since the attacker can observe the new network configuration only indirectly, he needs time to adapt his attack strategy.

We also investigate a novel defense mechanism against denial-of-service attacks based on a layer of proxy servers between the clients and the server. This defense can be set up quickly even after the attack has started.

Further we present a feasibility study of a worm, which performs denial-of-service attacks against cellular mobile networks. We show that overloading individual cells with user data is a serious threat if the attacker coordinates it correctly. A possible countermeasure, is a resource management system that moves users to less-loaded cells.

Resource Management in mobile Networks

Mobile Networks today consist of different access technologies. Besides traditional cellular networks like GSM and UMTS, modern mobile networks often incorporate wireless LAN hotspots. Today, usually the mobile phone or the user himself decides, which access technology to use. However, this is not optimal as only local knowledge is available for the decision. Network operators want to decide based on many parameters like radio conditions, network load and the user's demands, which user should be served by which access network. This could optimize the usage of the air-interface, which is a scarce resource.

For these complex decisions, measurement data has to be collected at the base stations and evaluated in the operator's core network. Unfortunately, the backhaul links that have to transport these mesaurements are another scarce resource.

In this thesis we introduce a publish/subscribe system which can efficiently collect and transport measurement data in mobile networks. Our system reduces data volume on the backhaul and allows flexible configuration of measurement jobs. We present simulation results that evaluate different measurement strategies. Our simulations also show the trade-off between the volume of the measurement data and the quality of the resulting handover-decisions.

Zusammenfassung

Ressourcenmanagement spielt in Technik und Wirtschaft eine wichtige Rolle. Allgemein versucht man dabei, die Ressourcen eines Unternehmens so effizient wie möglich zu nutzen, um einen (typischerweise finanziellen) Gewinn zu maximieren. In dieser Arbeit werden zwei spezielle Ressoucenmanagement-Probleme untersucht, die für Computernetzwerke relevant sind.

Ressourcenmanagement während Denial-of-Service Angriffen

Bei einem Denial-of-Service (DoS) Angriff versucht ein Angreifer, die Ressourcen von einem oder mehreren Internetdiensten aufzubrauchen, indem er große Mengen von Anfragen stellt. Neben dem Filtern des Angriffsverkehrs ist das so genannte Overprovisioning eine verbreitete Gegenmaßnahme. Das Opfer versucht dabei, genug Ressourcen zur Verfügung zu haben, um seine Dienste trotz des Angriffs funktionsfähig zu halten. Traditionelles Lastmanagement ist hierbei nicht ausreichend, da der Angreifer keine gutartige Last erzeugt, sondern strategisch agiert, um den Schaden zu maximieren.

Moderne Computernetze bestehen aus komplex miteinander verwobenen Diensten, mit vielfältigen Abhängigkeiten. Häufig werden Hilfsdienste wie DNS oder Datenbankserver benötigt, um die Benutzerdienste bereitzustellen. Solche Abhängigkeiten können Engpässe bei den Dienstkapazitäten erzeugen, die ein DoS-Angreifer ausnutzen kann, um mit wenig Aufwand viel Schaden anzurichten.

In dieser Arbeit entwickeln wir Methoden, um solche potenziellen Schwachstellen zu erkennen und dabei zu helfen, sie zu beheben. Dazu untersuchen wir spieltheoretische Modelle der Interaktion zwischen Angreifer und Verteidiger. Außerdem nutzen wir einen speziell zur Simulation von DoS-Angriffen entwickelten Netzwerksimulator in Verbindung mit Optimierungsalgorithmen, um gute Angreifer- und Verteidigerstrategien zu finden. Wir zeigen außerdem, dass die neue Technik der Virtualisierung einem Verteidiger während eines Angriffes Vorteile bringt, da sie es erlaubt, das Netz im laufenden Betrieb umzukonfigurieren. Der Angreifer kann diese Änderungen nur indirekt beobachten und benötigt daher Zeit, um seinen Angriff anzupassen.

Darüber hinaus untersuchen wir einen neuen Verteidigungsmechanismus, der auf Proxy-Servern basiert, die die Server von den Benutzern abschirmen. Diese Technik hat den Vorteil, dass sie auch während eines laufenden Angriffes schnell eingerichtet und aktiviert werden kann.

Außerdem untersuchen wir die potenzielle Bedrohung durch einen hypotethischen Wurm, der DoS-Angriffe gegen die Zellen eines Mobilfunknetzes durchführt. Wir zeigen, dass das Überladen einzelner Zellen mit Nutzerdaten eine ernsthafte Gefahr darstellt, sofern der Wurm entsprechend koordiniert wird. Eine Instanz, die Ressourcenmanagement betreibt, indem sie Benutzer in weniger belastete Zellen verschiebt, ist eine mögliche Gegenmaßnahme.

Ressourcenmanagement in Mobilfunknetzen

Mobilfunknetze bestehen heute häufig aus unterschiedlichen Zugangstechnologien. Neben traditionellen Zugangsnetzen wie GSM und UMTS gehören den Netzbetreibern häufig auch WLAN-Hotspots. Heute wird die Entscheidung über die aktive Netzkonnektivität meistens vom Endgerät oder vom Benutzer getroffen. Dies ist allerdings nicht optimal, da hierbei nur lokales Wissen zur Verfügung steht. Netzbetreiber würden gerne basierend auf vielen Parametern, wie Funkbedingungen, Netzauslastung und Nutzungsverhalten, entscheiden, welcher Benutzer von welcher Zugangstechnologie versorgt werden soll. Dies könnte die Nutzung der Luftschnittstelle optimieren.

Für solch komplexe Entscheidungen müssen Messdaten an den Basisstationen gesammelt und in das Kernnetz des Netzbetreibers transportiert werden. Leider sind die so genannten Backhaul-Links, die diese Daten transportieren müssen, eine weitere knappe Ressource.

In dieser Arbeit führen wir ein Publish/Subscribe System ein, das effizient Daten in Mobilfunknetzen sammeln und transportieren kann. Unser System reduziert das Datenvolumen auf den Backhaul-Links und lässt sich sehr flexibel konfigurieren. Wir evaluieren verschiedene Messstrategien und zeigen Simulationsergebnisse, die veranschaulichen, wie sich die Menge der Messdaten auf die Qualität der Handover-Entscheidungen auswirkt.

Acknowledgements

I believe, a doctoral thesis is not possible without the support of many people.

Ralph Holz and Liane Int-Veen both read the whole thesis and provided lots of valuable feedback. Christian Hoene and Heiko Niedermayer also shaped the thesis as it exists today. Prof. Georg Carle supervised this work and gave me the initial input upon which my ideas developed.

I also want to thank my parents and my girlfriend Renata for always supporting me, even in times when I was too busy with writing and could not care for them in the way they deserve it.

Many more people who contributed to this thesis are mentioned in the "Acknowledgements" sections of the individual Chapters.

Contents

1. Introduction

Resource management and resource allocation problems are common in many different contexts. In business economics, resources like equipment and human skill are assigned to projects for maximizing revenue. In investment, financial resources are allocated. In recent years it has become clear, that the resources that nature provides on earth are not endless and humanity has to make efforts to conserve them.

In computer science, resource allocation often refers to scheduling decisions. Resources like computing cycles of a CPU, memory or network bandwidth are always limited and in modern multitasking operating systems, different processes compete for these resources. In networked environments, such processes on a machine often act as servers, that answer requests from other machines.

Service providers who offer services — for example on the public Internet — need to make sure that their servers have sufficient resources to process the incoming requests from their customers. Large websites usually use server farms and load-balancers to assure this.

During a denial-of-service attack, an attacker tries to exhaust the resources that are available for a service by sending large numbers of requests. The attacker's goal is to make the service unavailable, it is overwhelmed with requests and can therefore no longer answer the requests of legitimate users. The service provider again can use resource management techniques like load balancing — either in advance or during the attack — to mitigate the attack's consequences. The difference of these scenarios to normal load-balancing is the presence of a strategic attacker who will exploit vulnerabilities. *Part I* and *Part II* of this thesis focus on resource management during denial-of-service attacks.

In mobile networks, the air-interface between the mobile device and the base station is the most scarce resource. Sophisticated algorithms care for scheduling decisions that not only depend on the desired Quality-of-Service and the current resource usage, but also on the radio conditions. Further, the backhaul-links that connect the base stations to the core network are another scarce resource — mobile network operators currently face the problem that the backhaul can not keep up with the increased data rates that technologies like HSPA and LTE offer. Introducing handovers between different radio access technologies — i.e. UMTS and WLAN — can improve the situation on the air interface but comes at the cost of requiring measurement data which has to be transmitted over the backhaul links. In *Part III* of this thesis, this trade-off is investigated.

1.1 Denial-of-Service Attacks

Denial-of-service attackers try to make services unavailable. The most common technique to achieve this is resource exhaustion. The attacker sends so many requests to the attacked systems that they do no longer respond to legitimate requests.

Such attacks first received public attention February 2000. A 15-yer old Canadian highschool student who called himself “Mafiaboy” attacked major new economy websites like Amazon, eBay and Yahoo, causing severe financial losses for these companies.

Today denial-of-service attacks are common on the Internet and seen as a major threat by Internet service providers ([1]). For example, several large-scale attacks with political backgrounds were conducted during the last few years. The victims of these attacks often use the term “cyberwar”, expressing that they regard these attacks as being equivalent to real-world attacks on their sovereignty.

Denial-of-service attacks can be seen as a resource management problem. If the owner of the attacked network distributes his resources in the right way, he can serve more requests and therefore make his systems more resilient against DoS attacks. In this aspect, DoS attacks are similar to the management of legitimate load, i.e. by using load balancers.

The difference to normal resource management is the presence of a strategic attacker who will exploit any weaknesses in the attacked network that he becomes aware of. For example an attacker may target a company’s DNS servers instead of the web servers — this attack strategy would bypass the load balancing mechanisms and have a similar effect as a direct attack since the provision of a web service depends on working domain name resolution.

Also seen from the attacker’s side, DoS attacks are a resource allocation problem. Even if the attacker has a large botnet of virus-infected PCs at his disposal, his resources are not infinite. The right resource allocation may be the deciding factor when the attacker and the victim are “equally strong”[1].

The possibility of using the same bots for spam distribution or phishing instead of denial-of-service creates costs of opportunity. Sending unnecessarily large amounts of traffic to a victim is also undesirable for the attacker as it increases the chance of the bots being discovered.

The goal of the denial-of-service part of this thesis is, to develop new methods to defend networks. Looking at the problem as strategic resource allocation, we can increase the effectiveness of the defender’s resource usage under DoS conditions. The defender can use the new techniques to find and fix vulnerable spots in his network.

[1]What equal strength means in this case will be defined in Section 5.2.

If services are placed in virtual machines, the defender can even re-configure his network during the attack.

To make sure that the investigated methods are in fact effective, we also have to look at the resource management decisions of the attacker. Often we will assume the attack to be carried out in a smart way, by an attacker who is aware of the defender's new possibilities. Sometimes we will even assume that the attacker has full knowledge of the attacked network, an assumption that would simply be unrealistic in reality. This is done because there is no point in developing a defense against an attacker that does not act optimally. When optimizing the defense against a weak attacker model, the result might have vulnerabilities that a real attacker can exploit. Our goal is not to simulate realistic attackers, but to find an strong defense — and to do this, strong attackers are necessary.

The final chapter of the denial-of-service part will study the possibility of denial-of-service attacks against mobile networks originating from mobile botnets. This scenario is unusual since until today, worms on mobile phones have not yet been very successful. Currently malware on mobile phones exists, but it rather spies on the user or tries to achieve a direct financial benefit for the attacker. DoS attacks by mobile botnets have not been discussed yet and if they happened today, mobile network operators would probably be as unprepared as Amazon and eBay were in February 2000.

1.2 Ressource Management in Future Mobile Networks

Today's mobile networks provide almost ubiquitous connectivity. In cellular network like the Global System for Mobile Communications (GSM) and the Universal Mobile Telecommunications System (UMTS), sophisticated algorithms decide which user should be served by which cell of the network.

Handovers between different access technologies, i.e. between UMTS and WLAN are a problem though. The decision how to serve a user best can be made locally close to the cells or even by the mobile device, but this is sub-optimal as only local information is available. It is advantageous when global knowledge about network load, radio conditions and user locations is available to assign the users optimally.

Often this knowledge is not available in today's mobile networks. For example performance monitoring of UMTS cells still relies on uploading files by FTP every few hours, there are no realtime capabilities. Also, transmitting additional measurement data from the cells to the core network creates load on the backhaul links between the cells and the core network and therefore introduces new costs.

With the introduction of the Long-Term Evolution (LTE) radio access technology, the data rates on the air-interface are increasing rapidly. The backhaul links can barely keep up with the new load and will be a major bottleneck during the next few years. Mobile network operators find it desirable to use the air-interface in more efficient ways by optimizing handovers, but they are not willing to sacrifice bandwidth on the backhaul for the required monitoring.

A goal of this thesis is the investigation of the trade-off between gains on the air-interface by using global knowledge for handovers and the required extra bandwidth on the backhaul for the necessary measurement data. We will further develop an efficient system for transmitting measurement data on the backhaul.

1.3 Structure of this Thesis

This thesis is structured as follows:

Part I provides some background information. Chapter 2 gives an introduction to denial-of-service attacks and the corresponding defense mechanisms. It also includes a brief history of notable attacks that happened on the Internet. Chapter 3 shows, how dependencies between services can be exploited by a DoS-attacker. Game theory is introduced in Chapter 4; the science of strategic interaction gives some insight into the attacker's and defender's options.

Part II models the interaction of attacker and defender during denial-of-service attacks. First, the Flones Network Simulator, a tool that was developed during the research for this thesis, is described in Chapter 6. In the following Chapter 7, Flones is used to harden networks against attacks by finding potential weaknesses which an attacker could expoit. Chapter 8 extends this work by allowing the defender to re-configure his network during the attack using virtualization.

Victims of denial-of-service attacks are often small businesses or even private persons. In many cases they are completely unprepared when an attack starts and use web-applications that are not designed to scale when adding additional servers. In Chapter 9 we investigate a proxy mechanism to defend such networks.

Finally Chapter 10 discusses denial-of-service threats which are caused by possible future botnets on mobile phones. A feasibility study for one specific attack scenario is conducted. Resource management by moving users to other cells is one possible countermeasure.

Part III continues at this point a different resource allocation problem. The goal is to improve the assignment of mobile devices to the cells of different radio access technologies, i.e. LTE, UMTS and WLAN, in future mobile networks. Collecting measurement data and transporting it from the base stations to the operators' core networks is necessary for this task.

In Chapter 12 a publish/subscribe system is developed, which can efficiently transport measurement data from the mobile base stations to the core network and therefore enables central decisions for heterogeneous handovers. In the following Chapter 13, a mobile network is simulated to determine the benefit and the costs of this mobility management solution.

Investigations in this thesis use different methodologies. If possible, models of the scenarios are solved analytically. However in most cases this is not feasible as the situations are too complex, therefore different simulations are the most common analysis method in this thesis. Where applicable and possible, models were verified using testbed experiments. An overview of the different Chapters' contents is given in Table 1.1.

Chapter Content	Software Description	Analytical Model	Simulation Results	Testbed Experiments
Part II: DoS Attacks as a Strategic Resource Allocation Problem				
Chapter 6 Description of the Flones Network Simulator	X			
Chapter 7 Detecting DoS-weaknesses in Networks		X	X	(X)
Chapter 8 Dynamic Defense using Virtual Machines			X	(X)
Chapter 9 A DoS Protection Scheme for Web Servers	X	X	X	(X)
Chapter 10 Study on DoS-attacks by Mobile Botnets			X	
Part III: Metering and Resource Management in Future Mobile Networks				
Chapter 12 Description of the Generic Metering Infrastructure	X		(X)	
Chapter 13 Simulation of GMI-based Handover Management	(X)		X	

Table 1.1: Overview of the thesis chapters.

Part I

Fundamentals and Background

2. Denial-of-Service Attacks

CERT CC, a computer security coordination center run by the US department of defense and Carnegie Mellon University, defines a *denial of service attack* (DoS attack) as *an explicit attempt by attackers to prevent legitimate users of a service from using that service*[1].

Generally DoS attacks are carried out over a network, the attacker has no physical access to the services. To render the service unusable, the attacker sends attack traffic which either overloads the destination (*flooding attack*) or uses protocol or implementation weaknesses to affect the destination (*semantic attack*) — or combines both approaches. The difference between semantic and flooding attacks is discussed in more detail in Section 2.2.2.

An example for a purely semantic attack is the classic "ping of death"[2], an implementation vulnerability that was present in many operating systems in the late 1990s. It allowed to crash a machine with a single fragmented ICMP packet that exceeded the maximum allowed IP packet size and therefore caused a buffer overflow. Similar vulnerabilities sometimes still appear today[3] and have to be handled by security-aware programming, timely publication and installation of security patches and also by firewalling unnecessary network access.

In this thesis we rather investigate "flooding" DoS attacks like the attacks on Amazon, eBay and other large web companies that took place in 2000 and 2001.

A history of such DoS attack will be presented in Section 2.1. Section 2.2 provides an overview of different types of DoS attacks, while Section 2.3 describes defense mechanisms. As novel DoS defenses are developed within this thesis, Section 2.4 focuses on how to evaluate them. Section 2.5 concludes this Chapter.

[1] `http://www.cert.org/tech_tips/denial_of_service.html`

[2] `http://insecure.org/sploits/ping-o-death.html`

[3] `http://www.h-online.com/security/news/item/One-false-ping-and-Solaris-is-in-a-panic-732224.html`

2.1 Notable DoS attacks

The 1997 PhD. Thesis of John D. Howard [2] appears to be the first scientific source that describes DoS attacks on the Internet. He predicted that DoS attacks might become a serious problem in the future, though they were not during the time of his study. He names the "Internet Worm" of 1988 which exhausted the resources of many network nodes as the first Internet DoS attack. According to CERT data, there were 143 reported DoS incidents in the time between 1989 and 1995, 63 of those belonged into the "process degradation" category that is closest to the definition of flooding attacks. All DoS attacks following the "Internet Worm" until 1995 were far less serious.

2.1.1 Early Denial-of-Service Attacks

In the late 1990s, DoS tools like Stacheldraht and Tribe Flood Network were developed[4]. Such tools allowed a 15 year old Canadian Student who called himself "Mafiaboy" to successfully attack various major websites, i.e. Amazon, Dell, Ebay and CNN in the year 2000[5,6]. These DoS attacks received a huge amount of public attention and can even today be regarded as the most spectacular attacks of this type.

In 2002 the DNS root servers were hit by a coordinated DoS attack[7] which affected all 13 servers but made only some of them unavailable. The DNS is a critical infrastructure and the DNS root servers can be considered the closest thing to a "single point of failure" of the Internet. They were attacked various times since then, however they are well-defended (see [3]). Until today, all attacks on the DNS root servers have had only very limited effects[8].

In the year 2003 computers infected with the W32/Blaster worm[9] launched a TCP SYN-Flood attack against `http://windowsupdate.com`. This attack can be considered a predecessor of modern botnet attacks, as it was performed by a large number of infected machines. However there was no direct control by a botmaster, but the attack's target and time were hard-coded in the worm. Even though the worm spread widely, the attack's effect was limited, as `windowsupdate.com` contained only a redirect to the real windows update server.

2.1.2 Botnet Attacks as a Business Model

Also starting in 2003 cyber criminals discovered DoS attacks as a business model. Web sites like the Malta-based online betting site "betfair" were attacked. The attackers intended to threaten the website owners to pay protection money for not attacking them[10,11].

In the following years, modern botnets appeared. Windows-hosts were infected by worms that turned the PCs into "zombies" (sometimes also called "drones") and connected them to a central command and control server. The so-called "bots" on

[4]`http://www.cert.org/archive/pdf/DoS_trends.pdf`
[5]`http://archives.cnn.com/2001/TECH/internet/02/08/ddos.anniversary.idg/index.html`
[6]`http://www.fbi.gov/libref/factsfigure/cybercrimes.htm`
[7]`http://c.root-servers.org/october21.txt`
[8]`http://www.heise.de/netze/meldung/Grossangriff-auf-DNS-Rootserver-143116.html` (German)
[9]`http://www.cert.org/advisories/CA-2003-20.html`
[10]`http://www.information-age.com/channels/security-and-continuity/it-case-studies/284366/how-to-survive-a-denial-of-service-attack.html`
[11]`http://www.csoonline.com/read/050105/extortion.html`

the infected PCs can be used for various purposes like sending unsolicited email, stealing private data from the infected system, and also for denial-of-service attacks of magnitudes that had been impossible before.

But botnet operators are usually thinking economically. If a denial-of-service attack brings no financial benefit, it creates costs of opportunity, since valuable resources (the bots) could have been used for more profitable actions — like sending Viagra Spam[12]. Further, public attention is bad for business, people might update their virus scanner and detect the bot if news about a botnet were in the mass media. These might be two reasons why botnet-attacks on highly visible targets are rare. In terms of resources a botnet with 9 million hosts[13] should be capable of successfully attacking almost any target on the Internet.

2.1.3 Denial-of-Service Attacks related to online Gaming

In 2009 there were news in mainstream media about private persons who used Microsoft's X-BOX live service and became victims to DoS attacks. It became known that botnet operators rented resources and know-how to players who lost games for taking revenge. However, already in 2001 and 2002 when the authors of [4] investigated DoS backscatter on the Internet, they found out that about 60% of the DoS victims in their data were dial-up and broadband users, often running IRC clients or multiplayer game clients like Blizzard's battle.net. They concluded:

> This experiment suggests that the majority of victims of the attacks we observed are home users and small businesses rather than larger corporations.

In early 2010 the video game publisher Ubisoft was attacked. At the time, Ubisoft had just introduced a new copy protection system for their games which required all players to have a permanent connection to Ubisoft's DRM servers while playing. This new method of protecting games from being illegally copied was highly controversial and intensively discussed in the media. When the first few games with the new copy protection were released, many customers were unable to play because of connection problems to the DRM servers. Ubisoft claims that this was caused by DoS attacks on their servers[14]. This was a major marketing disaster for the company.

Late 2010, the servers of the independent game project Minecraft[15] were attacked, while the game was still in an early stage of development. It is suspected that the attack was carried out by players that were not satisfied with the game's development pace, wanting to force new features — which in turn means that the attackers were probably teen agers with limited technical knowledge (script kiddies). As the attack was a SYN flood against which good defense techniques exist nowadays, full connectivity could be restored quickly.

2.1.4 Denial-of-Service Attacks due to political Conflicts

In recent years there were several DoS incidents that were related to political conflicts. In 2007 the country of Estonia came under serious attacks[16] while having

[12]`http://news.softpedia.com/news/Security-Expert-Analyzes-the-Botnet-Based-Economy-117634.shtml`

[13]`http://www.f-secure.com/weblog/archives/00001584.html`

[14]`http://www.eurogamer.net/articles/ubisoft-drm-was-attacked-at-weekend`

[15]`http://www.minecraft.net/`

[16]`http://asert.arbornetworks.com/2007/05/estonian-ddos-attacks-a-summary-to-date/`

political differences with the Russian government, in 2008 the same happened to Georgia[17]. Both victims claim that these attacks were acts of "cyberwar" by the Russian government while Russia denies any involvement. In 2009 South Korean and U.S. websites were victims to DoS attacks that were suspected to originate in North Korea[18]. In 2010 web-sites of Burmese exiles were attacked[19].

2.1.5 Denial-of-Service Attacks related to Wikileaks

In December 2010, the website Wikileaks was attacked several times after publishing secret correspondence between the US state department and US embassies in foreign countries. This problem could be solved since several hundred volunteers set up mirrors of the Wikileaks website.

In the second week of December, Wikileaks-sympathizers organized themselves via Facebook, Twitter and other web sites, to attack organizations that had recently acted against Wikileaks. Targets were companies who had stopped providing hosting services for Wikileaks (EveryDNS, Amazon) and financial companies who had frozen accounts with donations towards Wikileaks (the swiss bank Postfinance, Master Card, Visa and Paypal)[20].

The websites of Master Card, VISA and Postfinance were unavailable for multiple hours, apparently 720 attacking computers were enough to DoS `www.mastercard.com`. Nothing is known about the hardware configurations and access bandwidths of the attackers, but it can be guessed that many of the attacking nodes are home users with DSL lines.

Paypal was apparently attacked, but without significant effects. According to chat between attackers, they considered Amazon to be too strong to be attacked with 720 drones. As more volunteers participated, an attack on `amazon.com` was also scheduled, but could only slow the site down.

The attack utilized a number of different tools, including a Python-variant of the Slowloris attack tool, which is discussed in Section 2.2.2.

Especially interesting is the tool *LOIC*[21], an acronym for "Low Orbit Ion Cannon", a user-friendly program for window which allows to manually attack a target, or to set the program into a "hivemind"-mode, in which it connects to an IRC channel to receive orders. This way, the users voluntarily make their computers part of a botnet for participating in the DoS attack.

[17]`http://www.wired.com/dangerroom/2008/08/georgia-under-o/`

[18]`http://www.technewsworld.com/story/Suspicion-Centers-on-N-Korea-in-DoS-Blitz\-but-No-Smoking-Gun-67539.html`

[19]`http://www.irrawaddy.org/article.php?art_id=19558`

[20]`http://www.spiegel.de/netzwelt/web/0,1518,733520,00.html`

[21]`http://sourceforge.net/projects/loic/`

A quick test of LOIC 1.0.3 and the variant IRC-LOIC 1.1.1.14 (`https://github.com/NewEraCracker/LOIC`) against an Apache web server in a closed network revealed LOIC to be rather primitive. The tool can use three different attacks: UDP, TCP and HTTP. When selecting UDP or TCP, a number of sender threads is created that repeatedly send a user-defined text string — with UDP the text is written once into each packet, with TCP it is continuously written into the stream socket.

The HTTP mode did not appear to do anything with LOIC 1.0.3. Using IRC-LOIC 1.1.1.14, each thread opened a TCP connection and sent minimal HTTP-requests with only `GET` and `Host:` lines.

In all cases the target of the attack was not noticeably affected by an attack from a single machine (while with Slowloris, a single attacker can make an Apache web server unavailable). Also the attack requests have common properties that clearly distinguish them from legitimate traffic, which allows for filtering.

Generally, the software quality especially of LOIC 1.0.3 appears rather poor, the program sometimes did not stop attacks when the "stop" button was clicked and crashed several times during the test.

Another interesting fact is that the IRC servers that should coordinate the DoS attackers were themselves under DoS at certain times.

2.1.6 Summary

All these examples show that DoS attacks are still a major issue in Internet security. This is also supported by the 2009 Arbor Networks Infrastructure Security report ([1]), in which network operators named DoS attacks as the most significant anticipated operational threat for 2010.

This report reveals that DoS attacks are still frequent. Unfortunately the questioning methodology of Arbor networks makes it impossible to estimate interesting values like the average bandwidth of an attack. Among the asked ISPs, the largest observed DoS attack had a data rate of 49 gigabits per second, but most DoS attacks are far weaker.

2.2 Taxonomy of DoS Attacks

A good overview of DoS attack mechanisms is given in [5]. In this section only those distinguishing features of DoS attacks are discussed that are relevant for the following chapters.

2.2.1 Attacked Resources

A repeating theme of this work will be resource usage by denial-of-service attacks. All considered attacks share the property that they try to exhaust some resource at the victim.

Resources in this sense are:

- *Bandwidth* of a link in the network.
- Physical processing resources in some network host or router, i.e. *memory* (RAM, hard disk), *CPU* or *I/O bandwidth.*
- Limits due to software constraints (which might however be based on physical hardware constraints).

  An example of this category are "half-open TCP connections" that are created when a TCP SYN packet arrives. The number of such connections is limited — partially because of physical memory constraints but also because on the software side, the table that keeps track of such connections is too small.

We have to consider that an attack might involve multiple resources. For example a server might be running several services, some of which are CPU-intensive calculations while others require I/O bandwidth. The attacker will then have the option to focus his attack on a single service or to spread it among the services.

These victim-side resources are set in relation to the attacker's resources in Section 5.2.

2.2.2 Semantic vs. Brute-Force Attacks

As already discussed in the introduction to this chapter there are two flavors of denial-of-service attacks.

A *Flooding Attack / Brute-Force Attack* is a DoS attack that tries to exhaust a service's resources like processing power or link bandwidth by sending massive mounts of attack traffic to this service. The easiest example for flooding attacks is a simple UDP flood which consists of packets that are just designed to overload the network links.

On the other hand,a *Semantic Attack* consists of packets which have a semantic meaning to the destination. There is an almost endless amount of variants of semantic attacks, ranging from single packets that crash a router because of implementation weaknesses to requests that cause a server to perform a resource-consuming database search.

As pointed out in [5], many attacks do not fully fit into one of these two categories. Sending large amounts of requests to a server, each of which causes resource-intensive calculations is clearly a mix of both. Even a TCP SYN-Flood can be considered a semantic attack as it forces the destination to reserve resources.

Pure semantic attacks that exploit buffer overflow vulnerabilities are not considered in this work. All attacks that are in-scope need to have at least some flooding characteristic.

It is beneficial to distinguish finer categories of flooding attacks to describe the different attack vectors that the DoS defense mechanisms in the following sections will have to prove against:

1. Pure Flooding attacks do not require a connection setup. Every packet is independent of the others, there is no greater context. In this sense classic UDP and ICMP floods are "Layer 3 Flooding Attacks". These attacks do not specifically target protocol vulnerabilities but flood the target's network connection.

2. Attacks on TCP and IP exploit the specific semantics of these protocols. The SYN flood and attacks with fragmented packets belong to this type.

3. Attacks on Layer 7 exploit weaknesses in the application protocols. Exploiting a weakness on Layer 7 in many cases requires an established Layer 4 connection. Therefore the attacker has to behave according to the protocol specification of the Layer 4 protocol (TCP) at least during the connection setup. This also means that the attacker has to keep state at least for a limited time and that he can not use spoofed IP source addresses.

   An example is the so-called "Slowloris"-attack[22], a weakness in many HTTP servers regarding partial HTTP requests. When the attacker sends a part of an HTTP request, the web server already reserves a some memory to be able to process it quickly later on. Therefore sending larger numbers of such partial requests can make a web server unavailable. This attack is highly effective even at low data rates.

[22]`http://ha.ckers.org/slowloris/`

4. Attacks “above” Layer 7 again do not exploit protocol specifics. They are (at least initially) completely legitimate-looking requests that cause the server to process them normally. The number of requests (i.e. sent by many botnet-drones) and possibly also the choice of target tasks (i.e. requesting expensive searches instead of simple static web pages) make the attack destructive.

   This attack requires the attacker to follow the application protocol at least until the victim performs the resource intensive tasks that it should be tricked into doing. But as soon as it is advantageous, the attacker can deviate, e.g. to save the effort of having to close the connection.

In the Arbor Networks Worldwide Infrastructure Security Report of 2009 [1], Internet Service Providers were asked “what attack vector was employed for the largest attack they observed over the past 12 months ”.

The responses were:

- 45% Flood-Based (e.g., UDP, ICMP) *[Corresponding to our Category 1]*
- 23% Protocol-Based (e.g., RST, SYN, Frag) *[Corresponding to our Category 2]*
- 26% Application-Based (e.g., URL, GET, DNS, SQL) *[Corresponding to our Categories 3 and 4]* . As the most common targets, participants named DNS, SSH and HTTP.
- 6% other

This shows that all of these categories are a significant threat and worth investigating. According to the report, there is a shift towards the more economical Application Layer attacks. However pure flooding attacks have not become obsolete.

2.2.3 Source Address

The taxonomy in [5] distinguishes between spoofed and legitimate source IP addresses.

Spoofed IP addresses only allow for category 1 and 2 attacks in the above categorization. Given enough defender resources compared to the attacker resources, defense against these attacks is relatively easy by placing a layer of proxies between the clients and the actual web servers (compare Chapter 9). Such proxies require a complete 3-way TCP handshake before forwarding anything to the servers and therefore block any spoofed traffic.

The possibility of IP spoofing enables reflector attacks. With such attacks the attacker sends request packets to some other network node using the victim’s IP address as the source. Therefore the reply to the original packet is sent to the victim. If the reply is larger than the request, the attack is also amplified. There are variants of reflector attacks that use broadcast or multicast for the original request packet to trigger responses from multiple reflectors which further amplifies the attack.

According to Arbor Networks [1], even today IP spoofing is possible with most ISPs. The largest DoS attacks in terms of bandwidth are reflector attacks. Amplification can exceed a factor of 1:76.

2.2.4 Attack Rate Dynamics

The DoS taxonomy [5] also categorizes attacks by attack rate dynamics. Some DoS attacks apply strategies like a slow attack start or interchanging attacks from one of multiple sources to avoid detection.

This discussion is not relevant for most of this work, as we do not try to detect anomalies. However one interesting point should be considered. Attack rate dynamics usually refers to the amount of traffic sent, in terms of packet quantity or number of higher-layer requests. However, with attacks that take place above Layer 7, the type of request becomes interesting as well.

Displaying static web pages is not resources intensive, however often web sites include extended functionalities like a full-text search that require costly database queries.

An attacker would want to open persistent HTTP 1.1 connections to the server and then continuously send expensive requests. Legitimate customers would probably send a mixture of many cheap and few expensive requests. Such a situation can be used by the defender to detect clients that are behaving strangely. To avoid detection and countermeasures, the attacker would be forced to change his attack strategy to look similar to a legitimate client.

In Chapter 8 of this thesis attacker and defender will constantly change their strategies. However changes by the attacker do not have the goal of avoiding detection, but of optimizing the attack.

2.2.5 DoS or DDoS?

Generally most work regarding DoS attacks distinguishes between denial-of-service by a single traffic source and distibuted-denial-of-service (DDoS) attacks, where traffic originates at multiple sources. As botnets are common on today's Internet, we generally assume that the attacker has multiple systems at his disposal, but we will aggregate them back to a single attacker node when this is possible without loss of generality. As this work does not try to detect control traffic and worm spreading, the organization of the back end infrastructure is not relevant for most of this work (with the exception of Chapter 10).

We generally assume that the attacker has a certain amount of resources for his attack and that the number of actual drones is small enough do make it feasible to keep a list of their (real) IP addresses in memory if necessary. This is still realistic for a few thousand attacking systems while larger attacks are probably hopeless to defend against without having extreme amounts of resources.

2.3 DoS Defense Mechanisms

In the related work [5] a taxonomy of DoS defense mechanisms is provided. This taxonomy is not used for the discussion of related work here as most defense mechanisms were published later than the taxonomy, so some do not fit very well.

2.3.1 Overprovisioning

As DoS flooding attacks try to exhaust resources, the first defensive strategy is having more resources than the attacker can successfully exhaust. This can be achieved by adding mirror servers and load balancers. For example the DoS attack on Microsoft in 2001 that will serve as an example in Chapter 3 ended when Microsoft introduced additional DNS servers for their domains, which were hosted by Akamai. DoS attacks on the DNS root servers also usually fail or cause only minor damage because of overprovisioning ([3],[6]).

Large botnets have an overwhelming amount of available resources. However these resources can be used for other purposes than DoS attacks which bring the botnet operator a larger financial benefit (e.g. SPAM distribution). When assuming a rational attacker, we can also assume the resources of large botnets to be very limited: An attack with millions of bots will knock out any web server, but it also produces costs of opportunity for the attacker who could have used these bots in other financially attractive ways — so it will rarely be cost-effective.

2.3.2 Filtering

Filtering is a practically used method of defending against DoS attacks and also part of many publications related to DoS defense. The main idea is to introduce firewall-rules that block incoming attack traffic before it exhausts critical resources.

This may happen at different places in the network.

- In practice, even filtering on the attacked machine may help. The author of this thesis has talked to a server administrator who was attacked several times in 2009. The attack originated form a single source and targeted an application on the victim machine (an Apache web server). Filtering for the attacker's source IP address on the attacked machine helped as this prevented the attack traffic from reaching the vulnerable application.
- Filtering close to the attacked host is helpful as high-bandwidth links can still cope with the attack bandwidth, but the target machine may not be able to do so. It is also practically possible for operators of small-scale web services to ask their ISP for filtering (compare [7]). According to [1] ISPs take countermeasures when detecting DoS attacks, however there is no information available on detection rates and mitigation success.
- Filtering in foreign networks or even Internet backbones is a problem in practice, as different ISPs have to work together. Some DoS defense mechanisms that were published in the scientific community require some kind of cooperative filtering between autonomous systems (e.g. [8]).
- It is a common best-practice that egress filtering against IP spoofing should be done in all source networks. Even this is often not the case according to [1].

  The author of [9] suggests sophisticated DoS filters in all networks, to filter traffic close to the source. This would surely be effective, however it is a "tragedy of the commons" situation: ISPs have to invest money and build an infrastructure that protects not themselves but someone else. There is currently no incentive to do so.

Filtering is a building block of most DoS defense mechanisms.

The problem is distinguishing attack traffic from legitimate traffic. Filtering is very useful if the attack packets share a common property that can be used as a filtering rule, e.g. a common source address. As mentioned above, today there are still DoS attacks that originate from a single machine and use only a single kind of packets.

However, it is easy for an attacker to send different types of packets or disguise his traffic as legitimate requests. By using botnets and IP spoofing he can render source address filtering ineffective. The critical question therefore is, what to filter.

2.3.3 Traceback and Pushback

Traceback techniques, for example probabilistic packet marking ([10]) have been suggested to find the origin of spoofed IP packets. Traceback techniques would have to be introduced globally to be effective. As botnets have made DoS attacks with spoofed IP addresses obsolete, there is no more incentive to introduce a global traceback infrastructure.

Pushback on the other hand is an effort to filter attack traffic in routers before the target network. With pushback techniques, a router can tell his upstream router to drop packets matching certain criteria ([11]).

A problem of this approach is the size of the rule sets that would be required to filter DoS attacks from botnets (i.e. one rule per attacking IP). Further it remains questionable why a downstream node should be allowed to tell an upstream node to drop packets — if a pushback infrastructure existed, misconfigurations or attacks on this infrastructure could cause unpredictable damage. This leads to questions of responsibility. Who would pay for financial losses in such a case?

Like traceback, pushback would be most effective if deployed globally. Given the inherent risks of pushback techniques it is unlikely that this will ever happen.

2.3.4 Attack Detection

There has been a lot of research on detecting DoS attacks and using packet and flow-level characteristics to distinguish legitimate packets from attack packets.

Anomaly detection monitors traffic and calculates its characteristics, i.e. data volume or average flow lengths. It gives alerts when they change in an unexpected way. On the other hand, methods based on pattern matching try to identify known attack patterns in the data.

The authors of [12] propose a bandwidth-related heuristic to detect DoS attack flows. The authors of [13] use multiple techniques including spectral analysis to discover attack flows and tell if they originate from a single or multiple sources even if source addresses are spoofed. Automatic detection mechanisms are able to react to ongoing attacks rapidly.

Automatic detection systems can produce false negatives (in our case real attacks that are not detected) and false positives (legitimate traffic that is classified as attack traffic). Therefore it is dangerous to automatically generate filtering rules from the results. This argument becomes even worse given the presence of a strategic attacker who tries to exploit the internals of a detection system, either to remain undetected or even to trick the defense into dropping legitimate traffic.

2.3.5 Overlays and Proxies

A number of publications use proxies or even complete overlay networks to pre-filter traffic before it reaches the attacked servers. The proxy network has enough resources to serve as a first line of defense against the attack, e.g. because it consists of a huge number of nodes. In such approaches the proxies provide some possibility to distinguish attack traffic from legitimate traffic, which allows to filter the attack traffic.

In [14] and its extension [15], nodes of a chord-based overlay network act as proxies that authenticate the users. Legitimate packets are encapsulated and sent to the actual server. Routers between the server and the Internet filter traffic and allow only the encapsulated packets to pass. The authors of [16] also use filtering nodes between the clients and the server. These nodes may interact with the clients, sending them challenges that have to be passed before a request is routed to the server. In [17] filtering is done by encouraging legitimate clients to send more traffic during a denial-of-service attack. This should crowd out the attackers, as it is assumed that they already send traffic at the maximum possible rate.

Chapter 9 of this thesis investigates a novel proxy-based defense mechanism.

2.3.6 Proof-of-work and Captchas

It is often cheap for clients to send requests, while servers possibly have to invest much more effort in creating the response.

Proof-of-work techniques require the client to invest resources into a computational puzzle before the server does something expensive. Typical computational puzzles are brute-force attacks on weak cryptographic mechanisms ([18][19]).

For a web server such a defense can be implemented easily, by making the client perform the required calculations in JavaScript. Initially the client sends its request. The server replies with some information that has been encrypted using some weak key and a Javascript program that performs the brute-force attack. The client computes the solution and sends it back to the server who checks the response and answers with the desired content.

This method has a number of drawbacks. First of all, the server has to invest resources to accept client requests, generate puzzles and check for the validity of the solution. Such tasks should rather be outsourced to a network of proxy hosts. The proof-of-work tests also increase the response times for legitimate clients, therefore they should only be active during ongoing attacks.

The biggest drawback, however, is that the heterogeneity of Internet end devices is not considered. While a computational puzzle might be solved very quickly by a PC with a modern web browser, it may take ages to calculate on a mobile phone. Research approaches try to counter this by auction techniques, however, the core problem remains.

Captchas are “Completely Automated Public Turing tests to tell Computers and Humans Apart”. The most popular form are pictures of letters that have been distorted in some way. The idea is that the letters are still readable for humans while they are unrecognizable for OCR programs.

Captchas are commonly used on web sites where access by automated programs is undesired, i.e. when registering accounts or before a client downloads large amounts of data. Some websites initially assume that a user is a human, but if he behaves bot-like (i.e. by spidering the site) they present a Captcha. Like proof-of-work tests, Captchas can also be used as a lightweight authentication mechanism to prevent DoS attacks.

However, today Captchas do not accomplish what they promise. Most Captchas can be broken algorithmically, so they become machine-readable. On the other hand many Captchas are hard to read for humans. Websites with Captchas are often inaccessible for handicapped people, for example a blind person is unable to solve a visual Captcha.

Both, proof-of-work tests and Captchas can be used together with proxy nodes to reduce the effects of DoS attacks. They both do not work perfectly, but they will decrease the amount of attack traffic on the server, allowing to serve more legitimate requests. A related work is [16], which calculates scores from the results of different legitimacy tests and filters based on the results.

2.3.7 Capability-based Approaches

A different approach for DoS defense is based on so-called *capabilities* [20]. The key observation is that on the Internet anyone can send a packet to anyone else without obtaining a permission first — which makes DoS attacks possible. Capability-based approaches try to give control over the traffic to the receiver. The sender has to obtain permissions called *capabilities* for sending traffic from the receiver and possibly also from routers in intermediate autonomous systems on the path. Besides requiring major architectural changes to the Internet, a weakness of such schemes is that the mechanisms for granting these capabilities can themselves be vulnerable to DoS attacks (DoC — Denial-of-Capability Attack) [21].

The *Content-Centric Networks* (CCN) approach for a Future Internet [22] implicitly provides the same kind of protection. It is an interesting research question if Content-Centric Networks would be vulnerable to DoS attacks at all.

2.3.8 DoS Resistant Protocol Design

TCP is vulnerable to SYN Flood attacks as it only needs very few resources for a Client to send a SYN packet while the server has to reserve memory for TCP's state information. Large amounts of SYN packets can therefore exhaust the server's resources.

With SYN cookies ([23]) the server can delay the resource reservation until receiving a second packet from the client — and for sending a matching second packet the client also has to invest some resources. Therefore SYN cookies counter the TCP vulnerability.

The author of [24] investigates protocols for such DoS vulnerabilities. This is done by adding cost functions for the attacker and the defender to a model checker.

The Flones network simulator and the work presented in Chapter 7 of this thesis follow a similar idea. However not protocols are investigated, but network topologies.

2.4 Evaluation of DoS Defenses

In [25] a taxonomy of evaluation criteria for DoS defenses is given. This section gives a brief overview of those criteria. In Chapters 7, 8 and 9 where DoS defense mechanisms are discussed, this taxonomy will be used for evaluation as far as the categories are relevant to the given solution.

In this section the different subcategories of the taxonomy are described briefly as far as they are not self-explanatory.

- *Effectiveness*

  Effectiveness covers the general performance of the DoS defense mechanism. It is split into the following subcategories:

  - *Normal time:* How does the defense affect performance when there is no attack?
    - *QoS available for legitimate traffic:* e.g. delays, drop probabilities
    - *Resource Consumption:* e.g. CPU usage by cryptographic operations
  - *Attack time:* How well does the defense counter a DoS attack?
    - *QoS available for legitimate traffic*
    - *Amount of attack traffic allowed through*
    - *Resource Consumption:* e.g. processing power required by filters in routers

- *Ability to fulfill requirements*

  - *QoS requirements of legitimate applications:* e.g. web surfing or VoIP
  - *Legal requirements:* A certain robustness against DoS attacks might be required by external rules, e.g. rules for operating a DNS root server are defined in [3].

- *Robustness*

  Robustness refers to attacks against the DoS defense mechanism itself.

  - *Misusability:* Are there DoS attacks where the defense system makes things worse instead of better?
  - *Additional vulnerabilities created:* Is it possible to exploit new protocol mechanisms that were introduced as part of the DoS defense for DoS attacks? An example would be the injection of malicious filtering rules into a pushback mechanism.
  - *Resilience against changes in attack characteristics:* Is the system robust against changes to the attack traffic (e.g. if the attacker suddenly switches to IP spoofing)?

- *Configuration capabilities*

  Is it possible to re-configure the defense mechanism, e.g. when the attack changes or when it turns out that certain legitimate packets are classified as false positives and therefore dropped?

- *Dependence on other resources*

  This captures operational dependencies of the DoS defense system.

  - *Human intervention:* Is the system autonomous or does it require critical decisions to be met by a human?
  - *Technical resources:* Does the system depend on other technical systems? For example a filtering DoS defense depends on a mechanism to identify attack traffic.

- *Interoperability*

  How does the defense system fit into corporate infrastructures? Is interaction with other defense systems in other networks possible?

2.5 Conclusions

Denial-of-Service attacks first appeared in the 1980s and first caught public attention in the year 2000. Today they are even more threatening, as cyber-criminals have access to botnets of thousands of PCs.

It is interesting that major web sites are almost never affected. The fact that the web sites of Microsoft, Amazon, the White House and other large organizations who potentially have enemies on the Internet are always reachable shows that given enough resources, a defense against DoS attacks is possible.

Victims of DoS attacks are rather smaller companies, less-defended government websites and private persons. These organizations are usually unprepared when being attacked. The example of Minecraft in Section 2.1 shows that even outdated attacks like SYN-Floods initially work against these victims — at least for some time.

Key findings in Chapter 2:

- **Denial-of-Service Attacks are very common on the Internet, they were considered the most serious operational threat in a survey among ISPs.**
- **In 2009, the bandwidth of the strongest documented denial-of-service attack was 49 Gbit/s. However most attacks use far less bandwidth, the current trend is a shift from flooding to semantic attacks. For example the Slowloris attack works with partial HTTP requests and is very effective.**
- **Motivations of the attackers are extortion of businesses or the desire to make a political statement. Another common motivation — probably with younger script kiddies — is loosing in online games or being dissatisfied with the policies of computer game companies.**
- **By combining sufficient overprovisioning with filtering, a very effective denial-of-service defense can be achieved. Such methods make many potentially attractive targets like big company websites, government websites and critical Internet infrastructure like the DNS root servers robust against DoS attacks.**
- **Most successful DoS attacks target small businesses, less-defended government websites or private persons.**

3. Service Dependencies

One field that has not been investigated in previous work is the influence of service dependencies on denial of service attacks. The following real-world example shows, how service dependencies affect DoS attacks.

On January 25th and 26th 2001, the majority of Microsoft's services were made unavailable by a DoS attack[1] .

Figure 3.1 shows an approximation of what the company's network may have looked like at this time. The news articles about this topic are not very precise, but the illustration is sufficient to explain the main point.

Microsoft was running a number of different services, i.e. web sites and mail services. Most of these services relied on DNS name resolution for domain names like *microsoft.com* and *hotmail.com*.

At that point in time, the company had located all of their DNS servers in a single subnet which was connected to the Internet by a single router. Therefore this router was a single point of failure.

Due to a configuration problem the router failed on January 24th and Microsoft's services were unavailable for one day. This event was covered extensively by the press and technical news websites, therefore the existence of this single point of failure became known on the Internet.

When the router came back online, it became target of DoS attacks that interrupted Microsoft's services again at least temporarily during the next two days[2]. It is unknown, if the attack was carried out by a single attacker, it is possible that different individuals independently attacked the router after learning about the problem in the press.

[1] `http://www.networkworld.com/news/2001/0125mshacked.html`

[2] `http://www.heise.de/newsticker/meldung/Weitere-Attacke-auf-Microsofts-Websites-33586.html` (German)

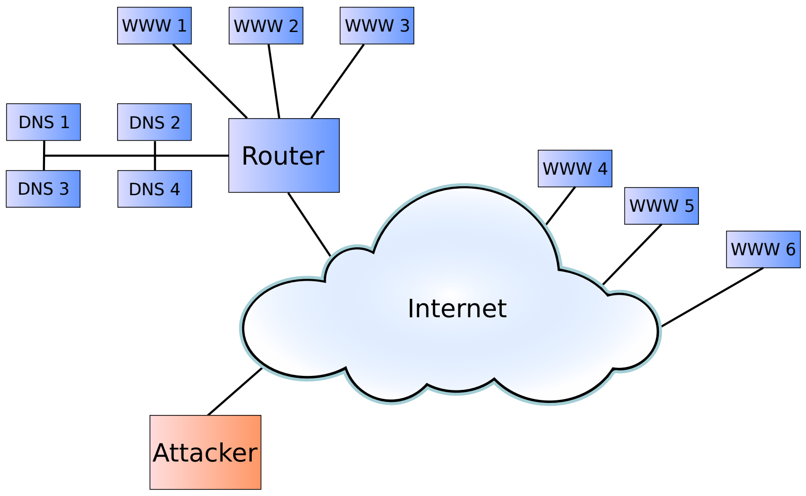


Figure 3.1: DoS attack against Microsoft in 2001.

Microsoft's denial-of-service problems were finally solved when they added new DNS servers in other networks with the help of Akamai.

When reading about this incident, Microsoft's mistake in the network configuration appears very obvious. However, in complex networks such misconfigurations can happen easily. Further it is interesting to investigate which general effects service dependencies can have on DoS attacks, even if the dependencies do not create a single point of failure. This problem space is investigated in Chapter 7.

3.1 Types of Service Dependencies

For this investigation it is useful to think about how services can interact with each other and which resources they share.

We consider three different types of service dependencies:

- *Client-Side direct dependency*: The client needs some specific information before being able to use a service. A common example is DNS, a web client has to resolve the IP address of the web server before being able to send a request.
- *Server-side direct dependency*: A service has to request some information before being able to answer a client's request. A common example is a web server that depends on a database server for serving dynamic content.

  This type of dependency comes in a large number of variants (i.e. a mail server depending on DNS or some authentication service). It is also the source of nested dependencies, as the dependency may have other dependencies of its own.
- *Router dependency*: Service requests depend on the routers on their path to be forwarded.

One important property of service dependencies in this context is that requests which originally seemed independent require some shared resource to be processed. The interaction of requests can take one of three forms:

- Requests (either user requests or requests caused by a dependency) are processed by the same service. In this case they have to share resources and also compete for places in the same queues.

  This work assumes that such requests end in a service-specific queue and are served in the same order as they arrived.

- Requests are answered by services that run on the same physical machine. There is some amount of isolation between the services, the operating system will try to schedule the services according to specific goals (i.e. fairness, priorities). The isolation is even stronger if the services are located in different virtual machines on the physical machine, which adds an extra layer of isolation.

- Requests have to pass through the same network element, i.e. a router or switch. They have to share network resources like bandwidth and router CPU. There might be some isolation if the router employs QoS mechanisms like IntServ or DiffServ, uses simple static packet prioritization or Layer 2 mechanisms like VLANs.

3.2 Detecting Service Dependencies

For the investigation method discussed in Chapter 7, it is necessary to know about service dependencies in the own network. There are several different methods available to collect and manage this information in large company networks.

Generally management of services and their dependencies is a task of network administrators. It can be fulfilled using tools that range from Excel Sheets to network management software like HP OpenView[26].

Networks can easily grow to a size and complexity that makes it very difficult to keep track of all service dependencies. There has been extensive research on discovering service dependencies from the network traffic [27, 28, 29, 30]. The method is usually to capture traffic at as many different locations as possibly (ideally in every host) and do reasoning on the observed packets. If a packet of type A arrives at a machine and shortly afterwards a packet of type B is sent from the same machine, there is the possibility that sending packet B was triggered by the arrival of packet A. If this happens multiple times, one can be increasingly sure that this sequence of events did not occur by accident, but that a real service dependency has been discovered. The actual analysis methods are more sophisticated, i.e. analyzing the frequency of events.

The author of this thesis also participated in developing an active approach to dependency discovery. The idea here is to cause load on a service S and measure the response time of a service T. If service T responds slower than normally, one can suspect that service T has a dependency on service S. To make the detection more accurate and to make multiple simultaneous probing processes possible, the load induced on service S is not constant, but follows a sinus pattern. This allows to detect possible dependencies in service T's response time by transforming the measured data into the frequency domain. Details can be found in [31] and [32].

3.3 Conclusions

In this Chapter we investigated the relation between service dependencies and denial-of-service attacks. We saw that Microsoft fell victim to a denial-of-service attack in the year 2000 and that this attack exploited a weakness in the company's network topology. We investigated different types of dependencies and discussed related work on detecting them.

Whether or not a network can be DoS-attacked not only depends on the amount of resources which are available for defending it. The network topology and service dependencies are key factors, as they can create weaknesses that an attacker can exploit to amplify his attack.

Key findings in Chapter 3:

- **Whether or not a network can be DoS-attacked not only depends on the amount of resources which are available for defending it. The network topology and service dependencies are key factors, as they can create weaknesses that an attacker can exploit to amplify his attack.**

4. Game Theory

Game theory is the science of strategic interaction. It is commonly regarded as a branch of applied mathematics, but also of economics as game-theoretic arguments play an important role in many economic questions.

Generally it can be said that direct application of game theory to complex real-world problems is usually not possible. Complete game-theoretic models even of simple situations tend to become very complicated. The Colonel Blotto Game, which is important for this thesis, is a good example, as its description is very simple while it took mathematicians 80 years to solve it. Still it is often useful to model simplified versions of strategic situations and try to extract results that are applicable in the real world.

Denial-of-service attacks are problems of strategic interaction. An attacker tries to do as much harm as possible while a defender attempts to prevent this. In Part II of this thesis, we will look at denial-of-service attacks as problems of strategic resource allocation. The Colonel Blotto Game, is a game-theoretic model of resource allocation, it is applicable to our denial-of-service scenarios and can be a source of valuable insights. The following Section 4.1 will give an introduction into a few basic game-theoretic concepts. Section 4.2 introduces the Colonel Blotto Game and investigates properties that are also relevant for denial-of-service attacks.

Good introductory books on game theory are [33] and [34].

4.1 Introduction

Game theory is split into many sub-fields like cooperative game theory, evolutionary game theory or auction theory. The following sections provide an introduction to non-cooperative game theory, the branch of game theory that deals with competitive and selfish strategic interaction between two or more individuals.

In this section we introduce the most simple definition of a game and the concept of a Nash equilibrium. More advanced game-theoretic concepts are only introduced when needed.

4.1.1 What is a Game?

Informally, a game in the game-theoretic sense is a situation where two or more players have to make strategic decisions. The different choices that each player has are called *strategies*. We denote the strategy space of player i (the set of all of player i's options) with the uppercase letter S_i, while a concrete strategy in this set is denoted by a lowercase letter $s_i \in S_i$.

At the end of the game, each player will get a payoff. This outcome is defined as a utility function[1] for each player. We will treat it as an abstract "score" that the players try to maximize. Each player's utility function depends on the strategy choices of all players.

With these concepts we can define a game as:

Definition 1. *A n-player game in normal-form is given as $G = (S_1, \ldots, S_n; u_1, \ldots, u_n)$ where*

- *S_i is the set of possible strategies for player i and*
- *u_i is the utility function of player i, $u_i : S_1 \times \ldots \times S_n \mapsto \mathbb{R}$.*

2-player normal-form games are usually drawn as a table:

	Player 2 Strategy C	Player 2 Strategy D
Player 1 Strategy A	$(u_1(A,C),\, u_2(A,C))$	$(u_1(A,D),\, u_2(A,D))$
Player 1 Strategy B	$(u_1(B,C),\, u_2(B,C))$	$(u_1(B,D),\, u_2(B,D))$

Here, the numbers in the brackets are the utility for player 1 and player 2 if the given strategy combination is played.

We illustrate the previous definition, with the following example.

Alfredo and Elisabetta are a couple and they want to go out together. Unfortunately, their mobile phones are broken, so they cannot agree on where to go. Therefore, both of them will just go somewhere and see if the other one is there.

It is the first common assumption for normal-form games that the game happens simultaneously and both players have to make their decisions without knowing what the other player will decide.

Alfredo wants to go to the casino, Elisabetta prefers the opera. However both of them will be very unhappy if they do not meet, they really want to spend the evening together.

We design a payoff matrix that fits to the story. Alfredo and Elisabetta will both get a positive score if they go to the same place. If they meet at the casino, Alfredo's

[1] In economy, the concept of *utility* is defined as *a number which describes the satisfaction that an individual experiences when consuming a good.* A *good* does not have to be a directly consumable item (e.g. an apple) but can also be far more abstract. For example we "consume" the security that police and fire-brigade provide. Further information can be found in textbooks on microeconomics.

score will be high while Elisabetta gets a high payoff at the opera[2]. If they go to different places, both of their scores will be zero.

	Alfredo: Gambling	Alfredo: Opera
Elisabetta: Gambling	(1, 2)	(0, 0)
Elisabetta: Opera	(0, 0)	(2, 1)

In normal-form games we also assume that the payoff-matrix is known to all players.

4.1.2 Nash Equilibria in Pure Strategies

Game Theory tries to determine, what the optimal choices of the different players in a game are. Here, one helpful concept is the Nash Equilibrium.

Definition 2. *A strategy profile $X^* = (s_1^*, \dots, s_n^*)$ is called Nash equilibrium, if for every player $i = 1, \dots, n$ and for every strategy $s_i \in S_i$*

$$u_i(s_1^*, \dots, s_{i-1}^*, s_i, s_{i+1}^*, \dots, s_n^*) \leq u_i(s_1^*, \dots, s_n^*) \tag{4.1}$$

In other words if all other players play their equilibrium strategies, player i has no incentive to deviate from his own equilibrium strategy. One can also say that each player's equilibrium strategy is a "best response" to all other player's equilibrium strategies.

In the example we have two obvious Nash equilibria: (Gambling, Gambling) and (Opera, Opera).

4.1.3 Nash Equilibria in Mixed Strategies

The original equilibrium concept by John Nash also allows a player to randomize between his strategies. If at least one player randomizes, we call the equilibrium a *Nash Equilibrium in mixed strategies.* On the other hand, if no player randomizes, we call it a *Nash Equilibrium in pure strategies.*

In our example the strategy profile

$$X^{mixed} = \left(\frac{1}{3}Gambling + \frac{2}{3}Opera,\ \frac{2}{3}Gambling + \frac{1}{3}Opera\right) \tag{4.2}$$

is a Nash equilibrium in mixed strategies. In this equilibrium the expected payoff[3] for both players is $\frac{2}{3}$.

[2]We do not assume scores to be comparable between Alfredo and Elisabetta. We also do not assume that a score of 2 means that a person likes something "twice as much" as a score of 1. The payoffs are just numbers that the players maximize, there is no deeper meaning involved.

[3]Calculation for Alfredo (he is player 2 here):

$$\begin{array}{rc} & u_{Alfredo}\left(\frac{1}{3}Gambling + \frac{2}{3}Opera, \frac{2}{3}Gambling + \frac{1}{3}Opera\right) \\ = & \frac{2}{9}u_{Alfredo}(Gambling, Gambling) + \frac{4}{9}u_{Alfredo}(Opera, Gambling) \\ & +\frac{1}{9}u_{Alfredo}(Gambling, Opera) + \frac{2}{9}u_{Alfredo}(Opera, Opera) \\ = & \frac{2}{9}\cdot 2 + \frac{4}{9}\cdot 0 + \frac{1}{9}\cdot 0 + \frac{2}{9}\cdot 1 \\ = & \frac{2}{3} \end{array}$$

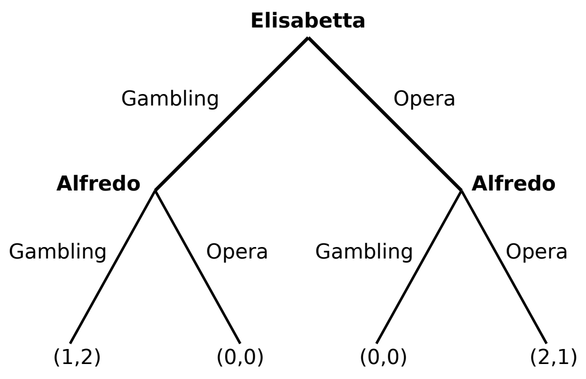


Figure 4.1: A sequential version of the battle of the sexes.

As John Nash has proven [35], it is sufficient to compare the mixed equilibrium strategy to all pure strategies to show that this actually is an equilibrium. Therefore in this concrete case it is sufficient to show that Elisabetta has no incentive to switch to one of her pure strategies as long as Alfredo sticks with his mixed strategy and vice versa.

The concept of mixed-strategy equilibria assures that each finite game has at least one Nash equilibrium. There are a number of games where randomizing makes sense intuitively, the simplest and also most prominent example is “Rock-Paper-Scissors”. However, in many cases — like in our example — mixed-strategy equilibria seem unnatural and counter-intuitive. When faced with a strategic problem, humans usually choose the solution that appears best to them — rolling dice in such a situation is rather uncommon.

4.1.4 Dynamic Games

In many real-live situations of strategic interaction, a player is able to observe the actions of his opponent before making his own decision.

Figure 4.1 shows a sequential version of the game “Battle of the Sexes” discussed above. Here Elisabetta can first decide where she wants to go. Alfredo is able to observe this decision.

Any subtree of this game is called a subgame. The corresponding equilibrium concept is the so-called *subgame-perfect equilibrium*. An equilibrium is called subgame-perfect if the equilibrium-strategies form a Nash-equilibrium in each subgame of the original game. Subgame-perfect equilibria are found by the technique of backward-induction, the game is solved from the end towards the beginning.

In his left subgame Alfredo would choose gambling to get a payoff of 2 instead of 0. In his right subgame Alfredo would decide for the opera to get a payoff of 1 instead of 0. Anticipating these decisions Elisabetta will choose to go to the opera. Therefore the subgame-perfect equilibrium of this game is (Opera, Opera).

This example shows that turning a simultaneous game into a sequential game can dramatically change the characteristics. Former equilibrium strategies have disappeared and the ordering has turned the symmetric game into one that highly favors one player.

Game theory has developed a number of additional concepts, i.e. for modeling incomplete information. However, as they are not used in the remaining part of this thesis, they are omitted here.

4.2 The Colonel Blotto Game

The Colonel Blotto Game is a well-known problem in game theory, which was first formulated by Borel in 1921 [36] and has been investigated by many mathematicians since. Even though the scenario is relatively simple, finding Nash equilibria in this game is hard and the results are often unintuitive. However, the Colonel Blotto Game is applicable to DoS attacks and is therefore investigated here.

4.2.1 Scenario

The commanders of two armed forces fight a war. Commander A has X_A armies at his disposal while Commander B is in charge of X_B armies. There is a limited number n of battlefields.

The basic Colonel Blotto Game lets both commanders choose a discrete distribution of their forces simultaneously. The commander who allocates more forces to a given battlefield j will win this battlefield. Each won battlefield gives a score of $+1$, while each loss is counted as -1, a draw gives a score of 0. The general with the higher score will win the war.

There is a large number of variants of the Colonel Blotto Game, e.g.

- continuous allocation of resources,
- different values of the battlefields,
- asymmetric resources,
- or sequential blotto games.

4.2.1.1 Equilibrium Example

Game-theoretic reasoning on the Colonel Blotto Game is very difficult. There are many equilibrium strategies in classic Blotto Games and they are all mixed-strategy and un-intuitive. The example equilibrium given here is taken from [37].

Assume $X_A = X_B = 10$ and $n = 5$. Define pure strategies $S1 \dots S5$ as follows:

Pure Strategy	BF 1	BF 2	BF 3	BF 4	BF 5
S1	4	3	2	1	0
S2	0	4	3	2	1
S3	1	0	4	3	2
S4	2	1	0	4	3
S5	3	2	1	0	4

In equilibrium, both players play each of these strategies with probability $\frac{1}{5}$.

The full set of equilibria for the continuous version of the game was finally found by Roberson in 2006 [38].

One central problem of game theory is that equilibrium strategies are often not convincing. Experimental game theory has shown that humans often choose strategies that were not predicted by game-theoretic concepts like the Nash Equilibrium. As equilibrium strategies for blotto games are not even intuitive (it is not possible to "see" why this is an equilibrium without calculations), it is unlikely that real-life individuals would follow the calculated strategies.

Therefore in the following sections, we will not argue based on these Nash equilibria but extract other useful results for our denial-of-service scenarios from the Colonel Blotto Game.

4.2.2 Blotto as a Zero-Sum Game

The colonel blotto game is a zero-sum game. There is only one score value, the payoff $u_B(s)$ for player B is $-u_A(s)$. Therefore, a special equilibrium defined by John von Neuman in 1928 is applicable here, the so called minimax solution.

In this solution player A has to decide, what strategy to play. It is common knowledge that it is the best strategy for Player B to minimize Player A's payoff. Therefore, Player A anticipates the worst-possible move by Player B and chooses his strategy in a way that minimizes the harm that Player B can do. John von Neumann's minimax theorem states about this situation:

Theorem 1. *In a finite two-player zero-sum game, with $u_B(s_A, s_B) = -u_A(s_A, s_B)$*

$$\max_{s_A} \min_{s_B} u_A(s_A, s_B) = \min_{s_B} \max_{s_A} u_A(s_A, s_B) \tag{4.3}$$

In computer science terms this means that the (inner) optimization loop of player B and the (outer) optimization loop of player A can be exchanged.

A minimax solution in mixed strategies is guaranteed to exist in all zero-sum games. This minimax solution is also a Nash equilibrium (the minimax solution is the stronger concept).

4.2.3 Non-existence of pure-strategy Nash Equilibria

The Colonel Blotto Game does not have pure-strategy equilibria. For simplicity, we prove this here for the case of continuously distributable resources.

Let $X_A = X_B$. Player A chooses the distribution of his forces $s_a = (a_1, \ldots, a_n)$, where n is the number of battlefields. Assume $a_i > 0$ for some i. A winning strategy for player B is as follows:

$$\begin{aligned} b_i &= 0 \\ b_k &= a_k + \frac{a_i}{n-1} \quad \forall k \neq i \end{aligned}$$

By not allocating anything to battlefield i, player B can win all other battlefields.

This is why two strategies s_a and s_b can never simultaneously be best responses to each other, assuming that s_a wins, the loosing player B rather would have wanted to play a strategy s'_b constructed from s_a as described above (intuitively, the situation can be considered the same as with Rock-Paper-Scissors; a game which does not have a pure-strategy equilibrium for the same reason).

4.2.4 Variants of the Colonel Blotto Game

Over the years, many variants of the Colonel Blotto Game have been published. Partly this was motivated by the difficulty of finding equilibria in the game, but partly also by the desire to apply the game to new real-life phenomena.

The authors of [39] investigate a sequential Colonel Blotto Game, but find an equilibrium which is not very convincing as the attacker has to make a choice in favor of the defender. In our DoS scenarios it will be more intuitive to model the attacker's benefit as the defender's loss, therefore a zero-sum model fits better. A very interesting work is presented in [40] which also investigates a sequential Colonel Blotto

Game and finds that there are circumstances under which the defender might prefer to leave locations undefended, even if he had spare resources to protect them.

In [37], the static Colonel Blotto Game is generalized. The authors investigate payoff functions that are different than the "the winner takes it all" function used in the classical Colonel Blotto Game and allow different payoffs if certain combinations of fronts are won. In Section 7.2 of this thesis we will introduce a specific payoff-function for modeling DoS attacks.

4.2.5 Consequences for Denial-of-Service Investigations

In its basic form, Colonel Blotto is a simultaneous game. When turning Blotto into a sequential game, the characteristics of the game totally change. The second player has a major advantage as he can observe his opponents choice and then follow the strategy outlined in Section 4.2.3. This way he will always win unless he has a significant disadvantage in resources.

Interpreting denial-of-service attacks as resource distribution decisions of attacker and defender, we see practical consequences of our game-theoretic investigations. Denial-of-service attacks are carried out by an attacker who is free to decide where to strike. He can change his attack strategy within moments by giving new orders to his drones. The defender on the other hand is static, changes to his network infrastructure require physical servers to be moved, new servers to be bought and new cables to be laid. Therefore the defender will loose when attacker and defender are about equally strong.

This fact was the inspiration for the work in Chapter 8 which tries to make the defender more agile.

4.3 Conclusions

In this chapter we have introduced Game Theory. In static games players act simultaneously, here Nash equilibria are the most common equilibrium concept. In mixed-strategy equilibria, the players randomize among their strategies.

If players act sequentially we have a dynamic game, in this case the subgame-perfect equilibrium is the easiest equilibrium concept.

The Colonel Blotto Game is a static game which can be applied to DoS attacks. However its equilibria are always mixed-strategy and not very intuitive. When modifying the Colonel Blotto Game to make the players act sequentially, the second player has a huge advantage if he can observe the first players's decision.

Key findings in Chapter 4:

- **A sequential version of the Colonel Blotto Game can be applied to the DoS problem. In this case the defender acts first, as he has to choose a static configuration for his network. The attacker can observe this configuration and attack accordingly.**
- **This gives the DoS attacker a huge advantage over the defender since he can exploit weaknesses in the defender's network. With about equal resources, the attacker will always win.**

Part II

DoS Attacks as a Problem of Strategic Resource Allocation

5. Overview

DoS attacks are considered a resource management problem in this thesis. Both, attacker and defender have a limited amount of resources (like CPU or network bandwidth) at their disposal. The defender's goal is using these resources to serve legitimate requests, while the attacker aims at preventing this.

We have seen the Colonel Blotto Game in Section 4.2, which is the game-theoretic equivalent to our situation. The main topic of Chapters 7 and 8 is the resource distribution of attacker and defender, which is similar to the Colonel Blotto scenario.

The defender has servers at his disposal, that can provide the services that he wants to offer. He can change the resource distribution, i.e. by upgrading servers, buying new servers or by changing the assignment of services to servers.

The attacker on the other hand has limited resources as well. As we saw in Chapter 2, denial-of-service attacks by large botnets with almost unlimited resources are rare. The much more common case is an attacker who controls a hand full of servers or has rented a small share of resources from a botnet. Therefore the attacker can not blindly throw packets at the attacked network, he has to carefully choose his target to assure the success of his attack.

In this thesis we make the simplifying assumption that the defender can not tell, which incoming requests are legitimate and which ones are part of the attack. Techniques like anomaly detection that can be used to automatically filter certain requests are not within the scope of this thesis (compare Section 2.3.2 and Section 2.3.4).

However filtering methods can be regarded as orthogonal to the methods developed in this thesis. If the defender can filter parts of the attack traffic before they consume scarce resources, only the remaining attack traffic becomes the defender's resource management problem. Further, load-management investigations that are presented in this thesis are also applicable to filtering systems for denial-of-service traffic.

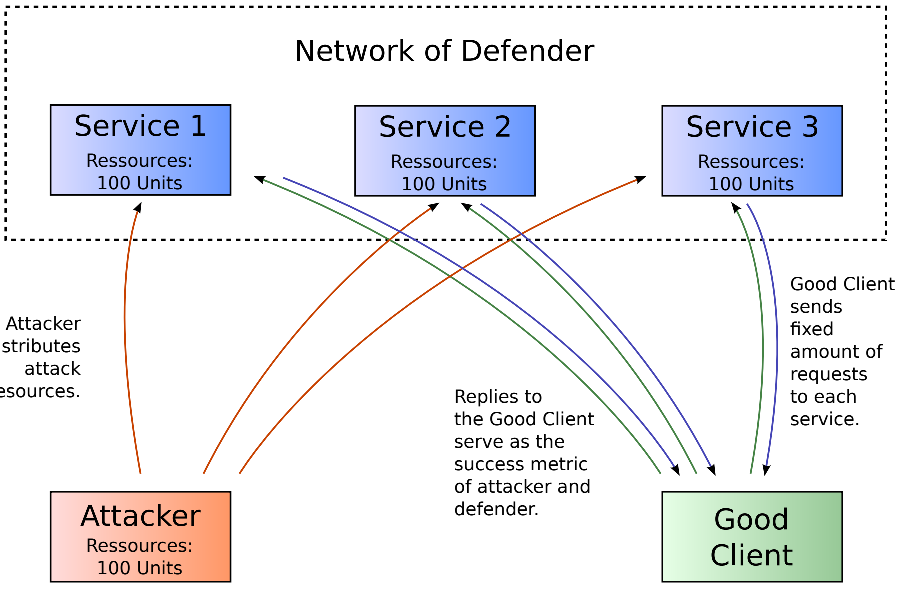

Network of Defender
Service 1
Ressources: 100 Units
Service 2
Ressources: 100 Units
Service 3
Ressources: 100 Units
Attacker distributes attack resources.
Replies to the Good Client serve as the success metric of attacker and defender.
Good Client sends fixed amount of requests to each service.
Attacker
Ressources: 100 Units
Good Client

Figure 5.1: The basic attack scenario.

Section 5.1 introduces the scenario and the players for the following chapters. Section 5.2 discusses, what resource usage means in our models. Finally Section 5.3 gives an overview of the chapters in Part II.

5.1 Scenario and Actors: Introducing Attacker, Defender and Good Client

Figure 5.1 shows the basic scenario of our investigations in Chapters 7 to 9. We have three actors:

- The *defender* owns the attacked network. He wants to provide a service to the good client, therefore his goal is to serve as many good client requests as possible. In the model he can not tell the difference between legitimate requests and attack requests, therefore he has to serve all requests.

  However, serving requests costs resources. If the resources at a specific service are exhausted, requests will be dropped. Main subject of this thesis is optimizing the defender's resource distribution to allow him to increase the amount of served requests.

  The defender's payoff in game-theoretic terms is the number of successfully served good client requests.

- The *attacker*'s goal is to harm the defender, in other words he minimizes the defender's payoff. To achieve this, he will use his own resources to send attack packets to the defender's services, trying to overload them. The attacker's payoff is equivalent to the dropped good client requests.

  The attacker is considered to be a single node here, the difference between normal denial-of-service attacks and distributed denial-of-service attacks is only represented by the attacker's amount of available resources.

  In some chapters we assume that the attacker is smart enough to know the defender's strategy or even the optimal attack strategy. This is legitimate as we want to optimize the defense and we can not do this assuming a unrealistically weak attacker.

- The *good client* is the aggregation of all legitimate users of the defender's services. He is assumed to be static for the analysis in this thesis. In real life, the load caused by legitimate users also changes during the day and sometimes in unexpected ways. This aspect is ignored for the time being. However by introducing load the good client can be seen to work in the attacker's favor.

  The number of replies that the good client receives from his requests to the defender's servers is used as the success metric of our DoS game.

5.2 Resources and the Relative Strength of Attacker and Defender

Real-life resources that a denial-of-service attack can exhaust have already been discussed in Section 2.2.1. They can include processing power, memory, network bandwidth, but also more high-level concepts like "half-open TCP connections".

In this thesis we will assume the resources of attacker and defender to be within the same order of magnitude. Denial-of-service games are within the scope of this thesis whenever the strategy matters: The attacker will win if he is smart and the defender chooses a bad strategy and vice versa.

This is not an unrealistic case, as we saw in Chapter 2. And even if the attacker was much stronger, the methods presented here are still helpful as they can be combined with other approaches like proxies and filtering. Defending against denial-of-service attacks by having enough resources is a highly relevant case as discussed in Section 2.3.1. Our work makes this defense more effective.

Mapping real-life resources to the investigated DoS situation is not easy, especially since attacker and defender do different things.

Assume a Core2-Duo with 2 cores and 3GHz HTTP-server and an identical machine which acts as a client. It is not clear that the server can serve the same amount of requests that the client can send, just because the machines are equally strong. Maybe the server (or the client) has to make expensive calculations or database-lookups to fulfill his tasks and therefore has to invest 100 times more CPU-cycles into a job than the client.

We will use abstract resource units that are scaled in a way which makes the model easier. The most common way of scaling will be that the defender needs one resource unit (i.e. of CPU) to serve the attack packets that the attacker has sent with one unit of the same resource. The actual numbers when building a simulation model from a real scenario have to be determined by measuring the performance of the participating nodes.

5.3 Outline of Part II

The following Chapters focus on the Subject of denial-of-service attacks in different scenarios.

- First in Chapter 6, the Flones Network Simulator is introduced. It will mainly be used for the analysis of network topologies in the following chapter.

- Making networks more robust against DoS attacks is subject of Chapter 7. We first introduce a theoretical model which shows, how attacker and defender will act in a very much simplified case. We extend the model by simulating scenarios where multiple services have to be distributed between multiple servers. Finally we investigate, how service dependencies create weaknesses that can be exploited by DoS attacks and test a method of optimally fixing these vulnerabilities.

- Chapter 8 is about further enhancing the defender's capabilities using live migration for DoS defense. Again a number of services are distributed among different servers, however, now the defender is able to move these services at any time — even during the attack. This is costly for the defender, but as the attacker has limited information about the new distribution, the attack will be less effective for a certain time.

- One DoS defense mechanism based on HTTP redirects is evaluated in Chapter 9. It is more transparent and standards-compliant than existing solutions, but its effectiveness is limited.

- Finally in Chapter 10, a possible denial-of-service attack on mobile networks is discussed. Here, a mobile botnet attacks the individual cells of the mobile network only, if enough attacking devices are present to actually cause harm.

6. The Flones Network Simulation Environment

When simulating denial-of-service attacks, traditional network simulators show deficiencies in two aspects. First of all, simulating large numbers of packets in event-based network simulators like NS2 and OMNeT takes a lot of time. Second, normal network simulators consider network bandwidth as the only relevant resource. Processing of packets inside the nodes happens instantly and uses no resources. Therefore simulations of application layer attacks, which mainly aim for exhausting processing resources at the victim, are not possible.

The Flones 2 network simulator was developed for denial-of-service investigations as presented in Chapter 7. The approach of Flones 2 is different from traditional discrete event network simulators in which an event is generated each time a single packet is received — which means that sending 100,000 packets over a single link causes 100,000 events. In Flones we wanted to aggregate events to greatly reduce the processing time.

Flones 2 as described here is the successor of the custom network simulator called Flones (**FL**ow **O**riented **NE**twork **S**imulator). Flones 1 had a number of nice conceptual ideas which are described in Appendix A, however the final software turned out to be relatively unpractical to use. Therefore the improved Flones 2 simulator was the basis of the actual research. Flones 2 will sometimes just be called Flones in the following, as Flones 1 is no longer in use.

Section 6.1 discusses related work. Section 6.2 names the goals of the Flones 2 software. Sections 6.3 and 6.4 describe the simulator, here Section 6.3.2 about the concept of a queue is the most important part for understanding the remaining chapters of this thesis. Section 6.5 gives some evaluation of Flones 2.

6.1 Related Work

The most common method of network simulation are so-called discrete event simulators. These simulators are based on a queue of future events. For example when a packet is sent, the delay of the link is calculated and an event is scheduled at the time, when the packet will arrive. The simulator processes the events one by one in the order given by the scheduled times. The running time of discrete event simulators linearly increases with the number of packets that have to be simulated. Common discrete event simulators are NS2[1], OMNet++[2] and Opnet[3].

Fluid network simulation models as presented in [41] express the packet flows analogous to flows of liquids. Complex protocols like TCP are modeled using differential equations. Time-continuous fluid simulation strongly abstracts over small changes in data rates and therefore looses accuracy. Therefore hybrid approaches like [42] have been suggested. Flones 2 is similar to that.

A unique new feature of Flones 2 is tracing of resources in each node. All traditional simulators consider network bandwidth to be the only resources, message processing in nodes happens instantly when the message arrives. Flones 2 introduces input queues, services that require resources like computing power for message processing, and a scheduler that assigns resources to the services.

6.2 Development Goals

The goals in the development of Flones 2 were:

- Instead of sending individual packets, Flones 2 should send message aggregates consisting of a message and a multiplier as the basic unit of data transmission. The simulation time should depend on the number of message aggregates, not on the number of messages contained in them. This speeds up simulations with large numbers of packets.
- Flones 2 should be able to simulate overload in routers and servers. This makes sure that not only pure flooding denial-of-service attacks, but also attacks on higher layers can be simulated.
- Flones 2 should allow the user to develop services in a natural way. Protocol implementations should be as similar as possible to handler functions that are used by real-world applications to react on incoming packets.
- There should be a strong relation between simulation rounds and real time, for example queuing delays should be included in the model.

6.3 Design

This section describes the design of Flones 2. The simulator works on Message Aggregates instead of individual messages, but transmits them similar to a normal network simulator. Flones 2 is round-based and assumes that all events that happen within a round occur simultaneously. In each round, Message Aggregates are taken from incoming links to the node's input queues. A scheduler decides, which messages can be processed by the node's services. Finally outgoing messages are sent to the outgoing links. These steps will be elaborated in the following.

[1] `http://www.isi.edu/nsnam/ns/`
[2] `http://www.omnetpp.org/`
[3] `http://www.opnet.com/`

6.3.1 Message Aggregates

MessageAggregates consist of one or multiple identical messages. Their attributes are shown in Figure 6.1. *singleSize* is the size of each message in the aggregate while *multiplier* is the number of messages.

singleSize and *multiplier* are defined as float variables that may take real values. This avoids problems that would otherwise arise from discretization. The calculations of the scheduler described in Section 6.3.3 would be much more complicated when working on integer values and desired properties like order-independence of *MessageAggregates* would have been hard to achieve.

content is the actual content of the message and consists of key-value-pairs where the key can be any hashable Python object and the value any kind of Python object. This field can also be used for convenience to save connection state that nodes assign to these packets. Therefore the content field is not directly related to *singleSize*.

MessageAggregate
source: *Node*
destination: *Node*
content: dictionary of arbitrary (key → value)-pairs
singleSize: float
multiplier: float

Table 6.1: Message Aggregate

6.3.2 Queue

The *MessageQueue* is the central class that defines the behavior of Flones 2. It consists of the *ma_roundlist*, a list of lists of *MessageAggregates*. Conceptually all *MessageAggregates* that arrive in the same round are assumed to have arrived simultaneously.

The basic operation of a *MessageQueue* is shown in Figure 6.1. In the beginning of each new round, a new empty list is created in and added to *ma_roundlist* (Step 1). *MessageQueues* are FIFO queues, so dequeueing will first retrieve the oldest messages (say the messages that arrived in round k).

Newly enqueued *MessageAggregates* are added to this most recent list (Step 2). Therefore the *MessageAggregates* in *ma_roundlist* always remain sorted by the round in which they arrived.

In Figure 6.1 the queue capacity is exceeded after the arrival of the new *MessageAggregates*. This is accepted for now, dropping of messages is the last step within a round. The reason for this is that there may be queues with a very high throughput compared to their capacity, therefore one always has to check the constraints after removing the *MessageAggregates* that leave in this round.

MessageAggregates are sent in Step 3. Here 350 messages can be sent, but the oldest *ma_roundlist* only contains 300 messages. Therefore those 300 messages from round one and another 50 from round two are sent. The 50 messages from round two are selected by taking an equal share from each *MessageAggregate*.

Equal *MessageAggregates* are merged before they are actually sent. Therefore in the example one *MessageAggregate* containing 110 Messages of Type A ist sent, one

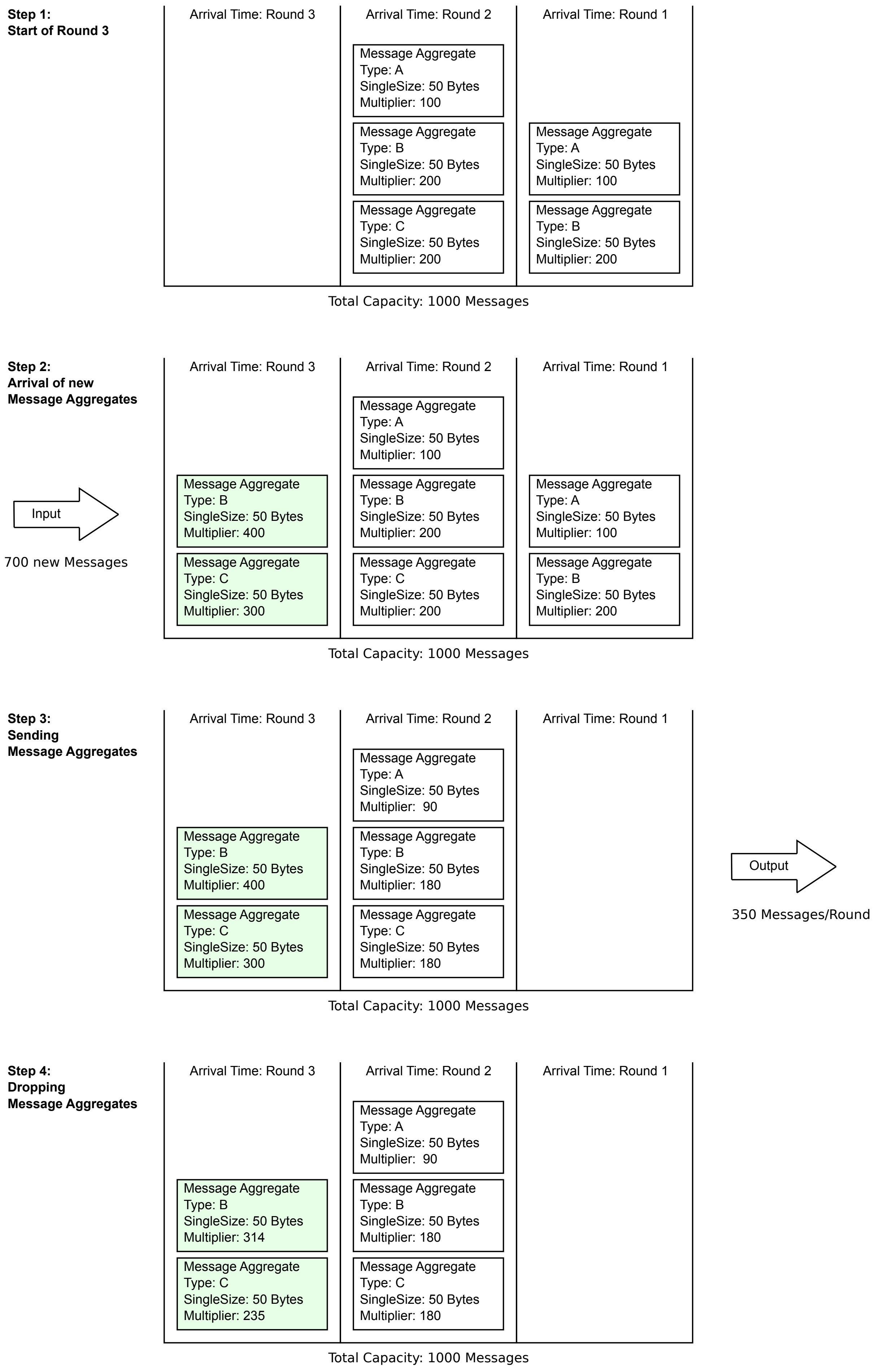

Step 1:
Start of Round 3
Arrival Time: Round 3
Arrival Time: Round 2
Arrival Time: Round 1
Message Aggregate Type: A SingleSize: 50 Bytes Multiplier: 100
Message Aggregate Type: B SingleSize: 50 Bytes Multiplier: 200
Message Aggregate Type: C SingleSize: 50 Bytes Multiplier: 200
Message Aggregate Type: A SingleSize: 50 Bytes Multiplier: 100
Message Aggregate Type: B SingleSize: 50 Bytes Multiplier: 200
Total Capacity: 1000 Messages
Step 2:
Arrival of new Message Aggregates
Input
700 new Messages
Arrival Time: Round 3
Arrival Time: Round 2
Arrival Time: Round 1
Message Aggregate Type: B SingleSize: 50 Bytes Multiplier: 400
Message Aggregate Type: C SingleSize: 50 Bytes Multiplier: 300
Message Aggregate Type: A SingleSize: 50 Bytes Multiplier: 100
Message Aggregate Type: B SingleSize: 50 Bytes Multiplier: 200
Message Aggregate Type: C SingleSize: 50 Bytes Multiplier: 200
Message Aggregate Type: A SingleSize: 50 Bytes Multiplier: 100
Message Aggregate Type: B SingleSize: 50 Bytes Multiplier: 200
Total Capacity: 1000 Messages
Step 3:
Sending Message Aggregates
Arrival Time: Round 3
Arrival Time: Round 2
Arrival Time: Round 1
Message Aggregate Type: B SingleSize: 50 Bytes Multiplier: 400
Message Aggregate Type: C SingleSize: 50 Bytes Multiplier: 300
Message Aggregate Type: A SingleSize: 50 Bytes Multiplier: 90
Message Aggregate Type: B SingleSize: 50 Bytes Multiplier: 180
Message Aggregate Type: C SingleSize: 50 Bytes Multiplier: 180
Output
350 Messages/Round
Total Capacity: 1000 Messages
Step 4:
Dropping Message Aggregates
Arrival Time: Round 3
Arrival Time: Round 2
Arrival Time: Round 1
Message Aggregate Type: B SingleSize: 50 Bytes Multiplier: 314
Message Aggregate Type: C SingleSize: 50 Bytes Multiplier: 235
Message Aggregate Type: A SingleSize: 50 Bytes Multiplier: 90
Message Aggregate Type: B SingleSize: 50 Bytes Multiplier: 180
Message Aggregate Type: C SingleSize: 50 Bytes Multiplier: 180
Total Capacity: 1000 Messages

Figure 6.1: Message Queue: Processing of Messages within one Round.

```
def do_HTTP_REQUEST(self, MA):
    """
    Handler for incoming HTTP requests.
    Sends HTTP replies back to the requests' source.
    """
    out_MA = MA.copy() # content, multiplier remain the same
    out_MA.source = MA.destination
    out_MA.destination = MA.source
    out_MA.size = 150
    out_MA.content['type'] = "REPLY"
    self.send_out(out_MA)
```

Figure 6.2: Example message handler method.

MessageAggregate contains 220 messages of Type B and the 3rd one contains 20 messages of Type C.

Finally in Step 4, the queue's capacity constraint is enforced. The queue currently contains 1150 messages, therefore 150 messages have to be dropped. Those are selected uniformly among the messages that arrived in the current round.

6.3.3 Service

Services are implemented in a quite natural way: Handler methods are defined for incoming *MessageAggregates*, as shown in Figure 6.2.

Each service also needs a method called *resource_request*. It receives a resource to be assigned (e.g. RAM) and a dictionary of already assigned resources (e.g. {CPU: 20 units, half_open_sockets:10}). The method returns the amount of resource units that the service wants to have in this round (e.g. 10 units of RAM). This method is called by the scheduler while distributing resources among the services.

Usually a service will look into its input queue to determine the right amount of resources to request. Assume that an example service needs one unit of CPU and one unit of RAM to fulfill a request. There are 20 pending requests in the input queue.

We assume that the scheduler calls:

```
resource_request(self, "CPU", {})
```

The example service will return **20**, because it needs 20 units of CPU to process its pending requests.

Further we assume that CPU is currently the bottleneck, the node's services request more CPU than there is available. In this case the second call by the scheduler might look like this:

```
resource_request(self, "RAM", {"CPU":10})
```

```
def resource_request(self, resource_name, assigned_resources):
    """
    The service returns the amount of resources it wants to have
    for this resource-type, given the assignment of other resources
    in assigned_resources.
    """
```

Figure 6.3: Header of the base class resource_request method.

The example service only got 10 units of CPU in this round, therefore it will not be able to process more than 10 requests from its input queue. As a consequence it will return **10** — there is no need to reserve more RAM as it will not be used anyway.

The exact way in which the resources are assigned is described in the following Section 6.3.4.

6.3.4 Scheduler

The scheduler assigns resources to each node's services in each round. While a normal scheduler in an operating system has to solve the question, which process will be the next to get access to a resource, the Flones 2 scheduler has to answer a different question: How will the fixed amount of resources that are available in one round be distributed among the services.

As a basic approach a proportional share scheduler, which assigns priorities to different services was chosen. The following requirements had to be fulfilled:

- *Handling of multiple resources:* The scheduler has to be aware of the fact that there are multiple resources and that a service needs to have all of his resource requirements fulfilled to process messages.
- *Efficient resource usage:* The scheduler must not assign resources to services that don't need them. The scheduler must not leave an amount of a resource unassigned if there still is a service with demand for this resource.
- *Possibility to prioritize:* It has to be possible to assign priorities to the different services. Services with higher priority are guaranteed to get a certain share of the total resources.
- *Fairness and order-independence:* The resources should be assigned equally to the services, i.e. two services that have equal priorities and demand equal amounts of resources should get equal assignments. The assignments must not depend on the order of services in the node's service list.

Fulfilling all of these requirements simultaneously requires looping over the services and resources several times in nested loops. Algorithm 1 gives an idea of how the assignment is done in the code.

In the following, the algorithm is described in a more prosaic way, starting with the innermost loop and adding the outer loops later on:

1. r is the first resource (e.g. CPU).
2. Ask each service, how much of resource r it wants to have.
3. All services requesting less than their share or exactly their guaranteed share of the available resources will get the full requested amount. A service's guaranteed share depends on his priority relative to the sum of all service's priorities.
4. All services requesting more than their guaranteed share of the available resources will get their guaranteed share for now.
5. Calculate how much of resource r is left.
6. *Inner loop — assign rest of r:* Return to Step 2 and distribute the rest of resource r among the services whose demand was not satisfied yet using the same scheme. Repeat until either resource r is completely spent or until there is no more demand for r from any services.
7. *Middle loop — assign the next resource:* Return to Step 2, continuing with the next resource. Whenever asking services about their desired resource usage, tell them which amount of resource r has already been assigned to them.
8. *Outer loop — only the assignment for the LAST resource is final:* A later resource limit might have caused a sub-optimal assignment of earlier resources. This is why we can only be sure that the last assigned resource is the correct value. Therefore we declare the assignment of the last resource r_{last} as final. We delete all non-final assignments and return to Step 2, starting again with the first resource.

To make the process more clear, here is an example of how the resource assignment works:

- Assume that we have two services that compete for two resources: CPU and I/O Bandwidth (IOBW).
- Service 1 needs 1 CPU and 2 IOBW to fulfill one request.
- Service 2 needs 2 CPU and 1 IOBW to fulfill one request.
- Service 1 has a priority of 3, Service 2 has priority 1.
- Each Service has 10 requests waiting in its queue. We have 10 units of each resource.
- *Inner loop for resource CPU:* Service 1 requests 10 CPU, Service 2 requests 20 CPU. Both services request more than their share, so Service 1 gets 7.5 CPU, service 2 gets 2.5 CPU. No unspent CPU resources are left.
- *Inner loop for resource IOBW:* Service 1 requests 15 IOBW because it got 7.5 CPU and needs twice as much IOBW to fulfill 7.5 requests. Service 2 only requests 1.25 IOBW for the same argument. In the first round of the inner loop, service 1 gets 7.5 and Service 2 gets 1.25 (it is below its share). In the second round Service 1 gets another 1.25 IOBW because there is no more competitor.

- Because Service one only got 8.75 IOBW, it can only fulfill 4.375 requests. Therefore it only requires 4.375 CPU, the rest would be wasted.
- *Outer loop:* We only accept the assignment of 8.75 IOBW for Service 1 and 1.25 IOBW to Service 2 as final. We return to re-assign the CPU.
- *Inner loop for resource CPU:* Service 1 requests 4.375 CPU and gets those. Service 2 requests 2.5 CPU and gets those. There is unspent CPU now, but we can not use it as we are limited by the IOBW.

Algorithm 1: The Scheduler assigns resources to the services.

```
input  : priorities
output : updated states, committed transitions
foreach $r \in resources, s \in services$ do
    $finally\_assigned\_resources(s, r) \leftarrow None$
// OUTER LOOP
for $\#resources$ do
    $temp\_assigned\_resources = finally\_assigned\_resources$
    // MIDDLE LOOP
    foreach $r \in (resources \setminus finally\_assigned\_resources)$ do
        // INNER LOOP
        while $unassigned\_resources(r) > 0$ AND $demand(r) > 0$ do
            foreach $s \in services$ do
                $share(s, r) \leftarrow unassigned\_resources(r) \cdot \frac{priority(s)}{\sum_{s1 \in services} priority(s1)}$
            foreach $s \in services$ do
                $demand(s, r) \leftarrow s.ask\_for\_demand(r, temp\_assigned\_resources(s))$
                if $demand(s, r) \leq share(s, r)$ then
                    $temp\_assigned\_resources(s, r) += demand(s, r)$
                else
                    $temp\_assigned\_resources(s, r) += share(s, r)$
            $calculate\_remaining\_unassigned\_resources(r)$
    $finally\_assigned\_resources(last\_resource) \leftarrow$
        $temp\_assigned\_resources$
```

Resources are split into two categories: Persistent and disposable resources. An example of a persistent resource is RAM, a node initially has a certain amount of it, services may reserve some RAM for their tasks. The total amount of RAM is fixed, so a service has to free RAM that it does not require any longer before any other service can use it. An example of a disposable resource is CPU — at the beginning of each new round, a certain amount of CPU-cycles is available. These are consumed by the services and disappear.

6.3.5 Node

The *Node* class in Flones 2 represents a node in the network. Figure 6.4 shows an overview of such a node.

Arriving *MessageAggregates* first have to pass the *Classifier*. This entity looks at the protocol-field in each *MessageAggregate* and distributes it to the assigned service. *MessageAggregates* that are not meant for this node are sent to the *ForwardingService*, a dummy-service that makes sure that *MessageAggregates* which are not

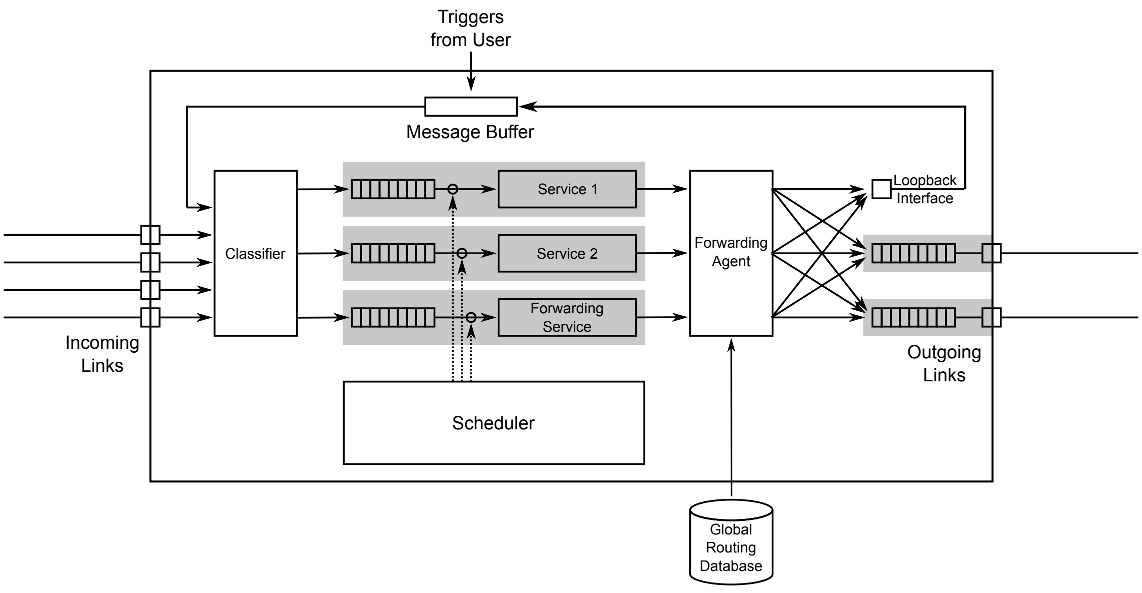


Figure 6.4: Nodes in Flones 2.

processed otherwise use up resources (the actual forwarding decision is taken by a separate *ForwardingAgent*). The *MessageAggregates* are enqueued at the Service's input queue and wait to be processed.

Next the *Scheduler* asks each service how much resources it wants to have. The service answers this question based on the fill level of its input queue. The *Scheduler* calculates the resource assignment based on the replies from all services and the amount of available resources as described in Section 6.3.4.

The services can now process incoming *MessageAggregates* with the assigned resources. During this process, new *MessageAggregates* might be created or it might be decided that existing ones have to be sent out again. Those *MessageAggregates* are passed to the *ForwardingAgent* which distributes them to an outgoing interface. The *MessageAggregates* are enqueued in the corresponding queue and sent out as soon as the traffic situation on the link permits it.

One special outgoing interface is the loopback interface. Messages on this interface are buffered in a *MessageBuffer* until the beginning of the next round, when they are passed to the classifier. This *MessageBuffer* can also be used to simulate triggers coming from the user.

This procedure shows one issue we have to be aware of: A message aggregate will at least take one round to pass a node. If the length of the simulation rounds is too long compared to the transmission delay of the links, this will increase the packet's travel times.

6.4 Additional Features

Flones 2 has a number of special features that were implemented because they were required at certain points in time. The following is a brief list of these functionalities:

- *Anycast:* Multiple Flones Nodes can be grouped into one anycast group. The effect is that these nodes are mapped to a single node when calculating the routing table.

- *Moving services at runtime:* Flones services can be moved between nodes. This can for example be used to simulate the migration of virtual machines. Costs for the migration are not automatically calculated, this has to be done manually in the code that triggers the service movement.

  A lookup mechanism has been implemented, which allows the user to set a destination service for a *MessageAggregate* instead of a destination node. The sending service on the source node has to make a lookup in a global service table to find the correct destination node for the messages. Like in reality it can happen that the destination service is moved while messages are on the way there, in this case those messages contain the old destination node and will therefore be sent to the wrong location.

- *Live Attacker and Live Defender:* Normally Flones 2 simulations are static, the defender has a static network and the attacker selects a fixed attack scheme prior to the simulation run. This mode of operations is used for iterative optimization of network topologies, results of this process are shown in Chapter 7.

  However there is a different mode of running more dynamic Flones 2 simulations. Between two simulation rounds, special agents for attacker and defender may re-configure the network at runtime. On the defender's side they would typically move services or resources, on the attacker's side they would change the attack's targets. This mode of operations has been introduced to be able to investigate the possibility of defending against DoS attacks by live-migrating virtual machines containing the services. This scenario is discussed in Chapter 8.

- *Multiple same-cost paths:* One important way to achieve resilience is redundancy. Flones 2 supports redundant paths to a destination and is also able to distribute traffic if multiple paths to the same destination have the same routing costs. Currently the sending router makes a local decision to distribute the *MessageAggregates* based on the interface's outgoing queue lengths.

- *Scenario Generation*: A scenario generator was implemented, which builds a topology based on the waxman algorithm ([43]). Then it randomly distributes resources and services with dependencies. The "random scenario" presented in Section 7.5 was initially generated by this method, but modified manually afterwards.

6.5 Correctness and Performance

Several tests were made, to make sure that Flones 2 performs as expected.

There were no direct comparisons to other network simulators like NS2 or OMNet. Pure forwarding tests of UDP packets are trivial, the correctness of Flones 2 in such scenarios could be assured using unit tests. More sophisticated simulations in traditional simulators usually involve TCP data flows, while Flones 2 currently does not contain a TCP model. It would be possible to build a model of TCPs congestion control based on TCP formulas for fluid simulation (like the one shown in [44]). On the other hand, NS2 and OMNet do not contain models of service's load behavior — one main feature of Flones 2.

The assumptions that the load model and the scheduler are based on have been verified in various tests. Practical tests have shown that the usage of CPU resources scales linearly with the load of a web server. The used amount of RAM depends on the load, but also on the number of worker processes. During the experiments that are presented in Section 8.2 it could be confirmed that in virtualized environments, the schedulers of hypervisors behave like the scheduler modeled in Section 6.3.4.

A number of plausibility tests was conducted on simulation scenarios, some of the results are presented in Section 7. Comparing simulation results to prior expectations sometimes yielded differences, however upon closer investigation it always turned out that the expectations were wrong while the Flones 2 model was correct. These differences were caused by not considering legitimate clients as a source of load during a DoS attack scenario and under-estimating the influences of dependencies.

Performance tests of Flones 2 were conducted as well. As expected — and in analogy to packet-level network simulators, the simulation time depends linearly on the number of message aggregate send events. In tests Flones 2 performed approximately 5000 of such send events per second on a 1.8 GHz machine, with tracing of events to a file being turned on. As expected, the runtime of Flones 2 also depends linearly on the number of rounds that are simulated.

6.6 Conclusions

In this chapter,the Flones 2 network simulator is described. Flones 2 aggregates individual packets and sends them in the form of a message aggregate. A special feature of Flones 2 compared with traditional network simulators is the calculation of resource usage in the nodes.

6.7 Acknowledgements

The author would like to thank Jörg Willmann, who participated in the development of the original Flones simulator and found the well-sounding acronym “Flones” during a brainstorming session. Clemens Plank wrote the first optimization scripts for it, Matthias Siegel and Andreas Korsten collected real-world data for the model and Alexander Gitter performed the plausibility tests and implemented several improvements.

7. Attacks against a Static Defender

Denial-of-Service attacks are a problem of strategic resource distribution. The defender usually has a network consisting of multiple servers. These servers host the services, some of which are accessible to end users while others are internal backend services.

The Microsoft DoS attack was already introduced in Chapter 3. It shows how an attacker can exploit weaknesses in the network infrastructure to cause huge damage with limited resources. In this example the weakness may seem very obvious. However, computer networks can become very complex, therefore, making sure that no such weakness exists is a difficult task.

In this chapter we investigate the case of a static network, which means that the defender can not make changes to the network when the attack has started. Buying and setting up hardware and laying wires takes time. For the attacker on the other hand, changing the attack strategy is a matter of giving a new command to his drones. He decides on a much shorter time scale.

We want to discuss the resource allocation choices of attacker and defender in this scenario, look at the influence of service dependencies and finally find and remove DoS weaknesses.

Further we want to investigate a simple statement regarding the attacker's strategy:

If the attacker has large amounts of resources (i.e. drones, bandwidth), he can afford to spread his attack among many target services. The more the attacker's resources are limited, the more he will have to focus the attack.

This chapter is organized as follows. After briefly discussing related work in Section 7.1, we will develop a game-theoretic model of DoS attacks in Section 7.2. In Section 7.3 the case of multiple services or virtual machines per physical server is discussed. In Sections 7.4 and 7.5, Flones 2 is used to find DoS-weaknesses in networks, while Section 7.6 investigates a method to fix them. Section 7.7 is about the missing pieces that are needed to develop a real-world DoS defense from the approach. Sections 7.8 to 7.10 conclude this chapter.

7.1 Related Work

Denial-of-service attacks and common defense mechanisms have already been covered in Chapter 2.3.

Game theory has already been applied to DoS investigations in the past. In [7] a DoS defense framework involving filtering in routers and firewalls is described and then evaluated game-theoretically. The authors of [18] look at DoS prevention techniques like computational puzzles and evaluate them using game theory. These game-theoretic considerations are very different from the resource-oriented view in this thesis. Service dependencies have not been considered in related work.

7.2 A DoS Model based on the Colonel Blotto Game

In this section we introduce a model for DoS attacks based on game theory. There are two active players, a DoS attacker and the owner of the attacked network, called defender in the following.

7.2.1 Resources and Payoff

We assume that the defender has n services. The defender can assign resources (i.e. CPU or network bandwidth) to the services which makes them harder to attack. The defender's strategic decision is the distribution of his C units of resources among his services. As a consequence, each service s_i will have a service capacity c_i, which denotes the number of requests that service s_i can handle.

The attacker also has a limited amount of resources at his disposal, which correspond to being able to send R requests. For the DoS attack, the attacker assigns requests to services of the defender. For service s_i the number of attack requests is $r_{a,i}$.

In analogy to game theory, the services can be seen as the battlefields of a war. Both players distribute their resources among different battlefields. Thus, our DoS attack model is a sequential version of the Colonel Blotto Game which was already introduced in Section 4.2.

To model the payoff functions of attacker and defender, we introduce a static third player, the good client. The good client is the representation of all legitimate users of the services and competes with the attacker for the services' resources.

The good client sends a representative amount of requests $r_{gc,i}$ to service i. If the service is overloaded, some of these requests will not be fulfilled. As discussed in Chapter 5, we assume that the defender can not distinguish legitimate and attack requests, both are treated equally. Therefore it is equally likely that an attacker or good client request is dropped. Thus, $\frac{c_i}{r_{gc,i}+r_{a,i}}$ is the fraction of all requests that can still be served. $r_{gc,i}\frac{c_i}{r_{gc,i}+r_{a,i}}$ requests of the good client will be served. We define the payoff for the attacker for attacking service s_i as the number of unanswered requests by the good client to the service due to overload.

$$p_{a,i}\left(r_{a,i}\right) = \begin{cases} 0 & \text{if } r_{gc,i} + r_{a,i} < c_i, \\ \left(1 - \frac{c_i}{r_{gc,i}+r_{a,i}}\right) \cdot r_{gc,i} & \text{otherwise} \end{cases} \tag{7.1}$$

The attacker's payoff from the whole attack strategy is

$$p_a(r_{a,1}, \ldots, r_{a,n}) = \sum_{i=1}^{n} p_{a,i}(r_{a,i}). \tag{7.2}$$

In our models we assume that the attacker can observe this metric, which is not the case in reality and therefore debatable. The rationale is as follows:

- Maximizing the number of dropped good client requests is the attacker's ultimate goal. Therefore this metric describes the attacker's payoff.
- The attacker can indirectly estimate the number of dropped good client packets from the observed loss of his own clients and his estimate of the popularity of the service.
- The goal of this analysis is hardening the network against the worst possible attack. Therefore it makes sense to assume global knowledge.

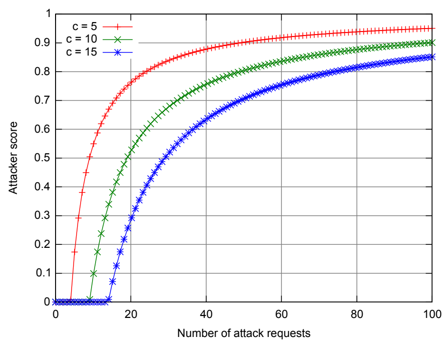


Figure 7.1: The payoff function for $r_{gc} = 1$.

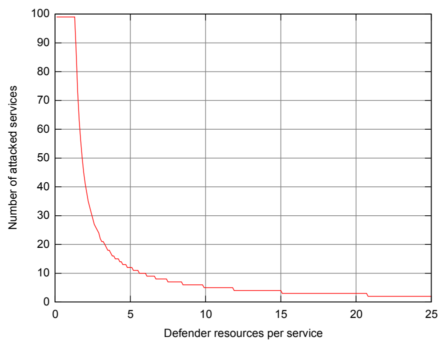


Figure 7.2: The number of attacked services for $n = 100$, $r_{gc} = 1$, $R = 100$.

7.2.2 Strategies of Attacker and Defender

We model the situation as a zero-sum game, with the defender's payoff being $-p_a$. The defender first has to decide, how to distribute his resources. After that the attacker distributes his own resources to attack the services. We solve the game using backwards induction starting with the second step (compare Section 4.1.4).

The attacker has to solve the maximization problem

$$\max_{r_{a,1},\ldots,r_{a,n}} \sum_{i=1}^{n} p_{a,i}(r_{a,i}). \tag{7.3}$$

For the further analysis of the model, we take some simplifying assumptions. The good client is sending an equal amount of requests r_{gc} to each of the services. To avoid degenerate cases in which either the attacker or the defender is not powerful enough to influence the game's outcome, we assume that the attacker is strong enough to attack a subset of the defender's services, but not all of them.

We make the following observations:

- The attacker has to choose how many services he will attack (how many of the $r_{a,i}$ will be larger than 0). As there is no payoff if $r_{gc} + r_{a,i} < c_i$, there is no point in attacking a service with less than $c_i - r_{gc}$ resources.
- $p_{a,i}$ as displayed in Figure 7.1 is concave as soon as $r_{a,i} > c_i - r_{gc}$. When investing more resources into attacking a specific service, the marginal payoff $\lim\limits_{\Delta r_{a,i} \to 0} \Delta p_{a,i}(\Delta r_{a,i})$ will decrease. Thus at some point it is better to attack one more service than to stick with the currently attacked ones.
- As can be seen in Figure 7.1, the attacker can achieve a higher payoff from the same number of attack requests, when focusing on weaker targets.
- The attacker will keep his marginal payoff equal among all attacked services. If the marginal utility at one place was higher, the attacker would be able to gain more by shifting resources there.

Summarizing we see that the best attacker strategy is to attack only a subset of the services. The services in this subset will be attacked in a way which equalizes the attacker's marginal payoff, so e.g. if all services are equally defended, they will also be equally attacked. In case of an unequal distribution of the defender's resources, the attacker will tend to attack the weaker-defended targets.

The defender has to distribute his resources C among the services and minimize the attacker's payoff by doing so. If the defender did not spread his resources equally, the attacker would strike the less defended services first. This would increase the attacker's payoff and the defender's resources would be wasted at a place which is not attacked. This points out that in this model and under the assumptions described above it is optimal for the defender to distribute his resources equally among the services.

Figure 7.2 was obtained by simulation using a Python script. Here the defender has 100 services, the vertical axis shows how each of them is defended. The good client causes a load of one resource unit on each of the services. The attacker has the capacity to create 100 resource units of load at his target services. As discussed above, the defender decides to spread his resources equally among the services, the attacker chooses a subset of services to attack with equal strengths. Figure 7.2 shows the number of services that the attacker decides to attack on the vertical axis, while increasing the defender's resources on the horizontal axis. One can see the anticipated behavior, the attacker has to concentrate his resources when attacking more strongly defended targets.

7.2.3 Experimental Verification

The above considerations were verified in a testbed. The server was a 3 GHz Core2 Duo machine, running Linux, an Apache web server and PHP5. As a purely CPU-bound service, a PHP script that factored a large number was used. A single computation of this task took roughly 7 milliseconds.

The Attacker was a Core2 Quad with 2.4 GHz, the Good Client a Core2 Duo with 2.5 GHz. Both, attacker and Good Client are realized using Apache Benchmark. All three computers resided in the same local area network.

In the real world, the situation is different than in the theoretical model. In theory we assumed that Attacker and Good Client send fixed amounts of requests, when the capacity of the server is exceeded the extra requests are dropped.

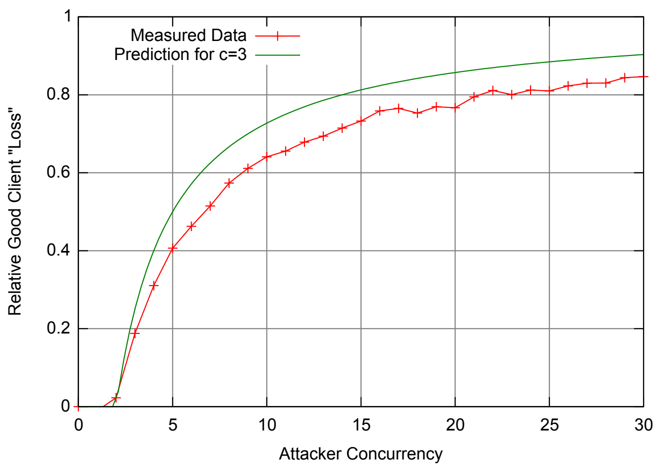


Figure 7.3: Comparison of a testbed Experiment with the theoretical considerations. The axis labels have been chosen to make the curve comparable to Figure 7.1. A value of zero on the vertical axis means that the server serves the Good Client optimally, while increasing values show the Good Client's loss due to the server also having to serve the attacker.

In reality, we have to set-up a TCP connection before being able to send requests. If multiple requests are to be sent, existing TCP connections are usually re-used. Since TCP includes error-handling, usually no requests are lost.

Apache spawns worker processes to handle requests. By default up to 200 processes can be running simultaneously and another 100 TCP connections can be waiting in a queue until they are assigned to a worker process.

On the client side, Apache Benchmark has a "concurrency" parameter that determines, how aggressive the benchmark will be. This parameter sets the number of simultaneous open TCP connections. In each connection, the client will send a single request and wait for the reply (stop and wait). Apache Benchmark as configured for this experiment sends requests for a fixed time and counts the number of replies. Requests that have been sent before the end of the experiment but have not received a reply when the time has ended are not counted as failed requests, but simply ignored, which is a difference to the Good Client that will be used in Section 9.6. This difference is negligible, however, as there is only a single active Good Client request at any time compared to 4269 Good Client requests that are successfully served under optimal conditions.

Because of this, almost no request actually fails. In the following experiment, the number of failed Good Client requests is zero. However, the attacker "steals" processing power that would otherwise be used to serve the Good Client. As the Good Client sends his requests one by one and as each request takes longer to process, the number of responses that the Good Client receives gets lower. In other words, even though the setting is different, we still expect to see the predicted effect.

Figure 7.3 shows exactly this behavior. Since the server is a dual-core machine, we would expect it to be able to serve two requests (i.e. one from the Attacker and one from the Good Client) simultaneously. However, each request spends some time on the network and in the network stacks of the participating machines. The experimental data shows that the Good Clients is still served optimally at a server-side concurrency of 3. Therefore the theoretical estimate for $c = 3$ and $r_{gc} = 1$ has been drawn in the Figure as a reference.

This experiment shows that despite the many complicated aspects of real-world servers, the model presented here behaves very similar to real systems. However some parameters have to be interpreted differently. Sections 8.2 and 9.6 contain additional testbed experiments.

7.3 Multiple Services per Server

In reality, often multiple services — for example web sites — are run on the same machine. This section focusses on calculating optimal assignments of services to servers. This can be seen as a preparation for Chapter 8, where the defender will gain the possibility to move the services during the attack.

The following results were obtained using a similar model as in Section 7.2. A scheduler was added which is a version of the scheduler in Section 6.3.4, but limited to a single resource. We will take the scheduler as given, scheduling is not part of the defender's strategy.

Again we will calculate time in rounds, therefore our scheduling decision is not the same as in a regular operating system. We do not decide which service should be run next, but which amount of resources each service will get during the next round of time.

In the simple case of a single resource this scheduler will act as follows:

- Each service has a priority p_s. The share of service s is given as $\frac{p_s}{\sum_{i \in S} p_i}$.
- A service requesting less resources than its share will get the requested amount of resources. The remaining (not needed) resources will be collected in a pool.
- A service requesting more than its share will get its share.
- This procedure will continue recursively. In each pass the resources from the pool will be distributed among all services who require more.

Attacker and defender strategies were calculated using optimization algorithms to find minimax-sound strategy combinations. For simplicity, all services have equal scheduling priorities and the same amount of good client traffic.

Figure 7.4 shows the optimal strategies of attacker and defender in an example scenario. The defender owns three servers with the capacity to serve 100 requests per time unit each. Five services are run on these servers, the good client causes a load of 10 requests on each of these services. The attacker's capacity is increased from top to bottom. As in the previous sections, the attacker's score is based on the number of lost good client requests, therefore the maximum achievable attacker score is 50. The results of attacker and defender optimization are optimal, but not unique (it is possible to swap services for getting a different assignment with equal score).

We see that the defender tries to distribute his services as far as possible. This is a logical choice as a concentration of services on the same server would also concentrate good client requests — increasing the initial server load and allowing for a higher attacker score as there are more good client requests that can potentially be dropped.

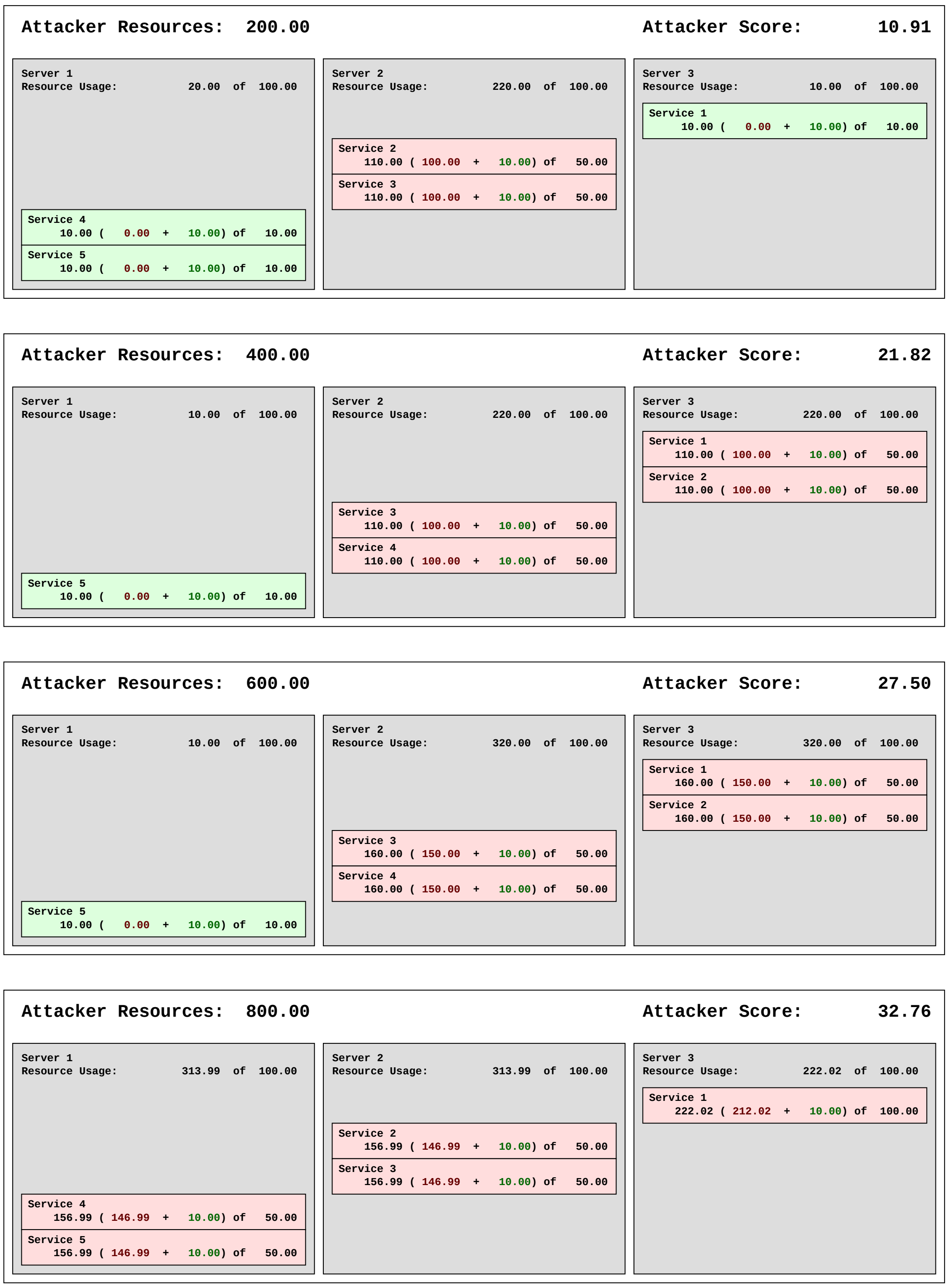

Attacker Resources: 200.00
Attacker Score: 10.91
Server 1
Resource Usage: 20.00 of 100.00
Service 4
10.00 (0.00 + 10.00) of 10.00
Service 5
10.00 (0.00 + 10.00) of 10.00
Server 2
Resource Usage: 220.00 of 100.00
Service 2
110.00 (100.00 + 10.00) of 50.00
Service 3
110.00 (100.00 + 10.00) of 50.00
Server 3
Resource Usage: 10.00 of 100.00
Service 1
10.00 (0.00 + 10.00) of 10.00
Attacker Resources: 400.00
Attacker Score: 21.82
Server 1
Resource Usage: 10.00 of 100.00
Service 5
10.00 (0.00 + 10.00) of 10.00
Server 2
Resource Usage: 220.00 of 100.00
Service 3
110.00 (100.00 + 10.00) of 50.00
Service 4
110.00 (100.00 + 10.00) of 50.00
Server 3
Resource Usage: 220.00 of 100.00
Service 1
110.00 (100.00 + 10.00) of 50.00
Service 2
110.00 (100.00 + 10.00) of 50.00
Attacker Resources: 600.00
Attacker Score: 27.50
Server 1
Resource Usage: 10.00 of 100.00
Service 5
10.00 (0.00 + 10.00) of 10.00
Server 2
Resource Usage: 320.00 of 100.00
Service 3
160.00 (150.00 + 10.00) of 50.00
Service 4
160.00 (150.00 + 10.00) of 50.00
Server 3
Resource Usage: 320.00 of 100.00
Service 1
160.00 (150.00 + 10.00) of 50.00
Service 2
160.00 (150.00 + 10.00) of 50.00
Attacker Resources: 800.00
Attacker Score: 32.76
Server 1
Resource Usage: 313.99 of 100.00
Service 4
156.99 (146.99 + 10.00) of 50.00
Service 5
156.99 (146.99 + 10.00) of 50.00
Server 2
Resource Usage: 313.99 of 100.00
Service 2
156.99 (146.99 + 10.00) of 50.00
Service 3
156.99 (146.99 + 10.00) of 50.00
Server 3
Resource Usage: 222.02 of 100.00
Service 1
222.02 (212.02 + 10.00) of 100.00

Figure 7.4: Optimal attacker and defender strategies with equal servers for attackers with 200, 400, 600 and 800 resource units.

The attacker has to make the same decision as before, he has to decide how many servers to attack. If the services on one server are equally valuable (in terms of good client requests) and equally prioritized (by the scheduler), he will always attack them equally. With increasing resources the attacker will attack more servers as he did in Section 7.2.

For Figure 7.5, the resources of the first server were doubled. The defender chooses to keep three services on the stronger machine, as this increases the minimum resources available to all services. Again the argument is the same as in Section 7.2.

When the attacker is weak, he will attack one of the weaker servers with a single service. However as soon as he is strong enough, he will switch to attacking the strong machine for maximizing the number of dropped good client requests.

7.4 Finding DoS-Weaknesses using Flones

In this section we switch from the simplistic game theoretic model to a network simulation. Given a network topology, servers and services and information about service dependencies (compare Chapter 3), we want to calculate the optimal attack. This allows the defender to assess potential DoS vulnerabilities of his network infrastructure.

The method is based on the Flones 2 network simulator. Conceptually it would be possible to replace the simulator component with a traditional packet-based simulator like NS2, however Flones 2 has two major advantages:

- The Flones 2 model of MessageAggregates makes the simulations fast.
- In Flones 2 a scheduler makes sure that resources like CPU and RAM are required for message processing (see Section 6.3.4). If there are insufficient resources, messages remain in the service's queues which will eventually overflow. In traditional network simulators, the only "resource" is the capacity of links, processing of messages happens immediately and without costs.

  Therefore unless a DoS simulation focuses on pure flooding attacking attacks, a way to handle other resources would have to be introduced.

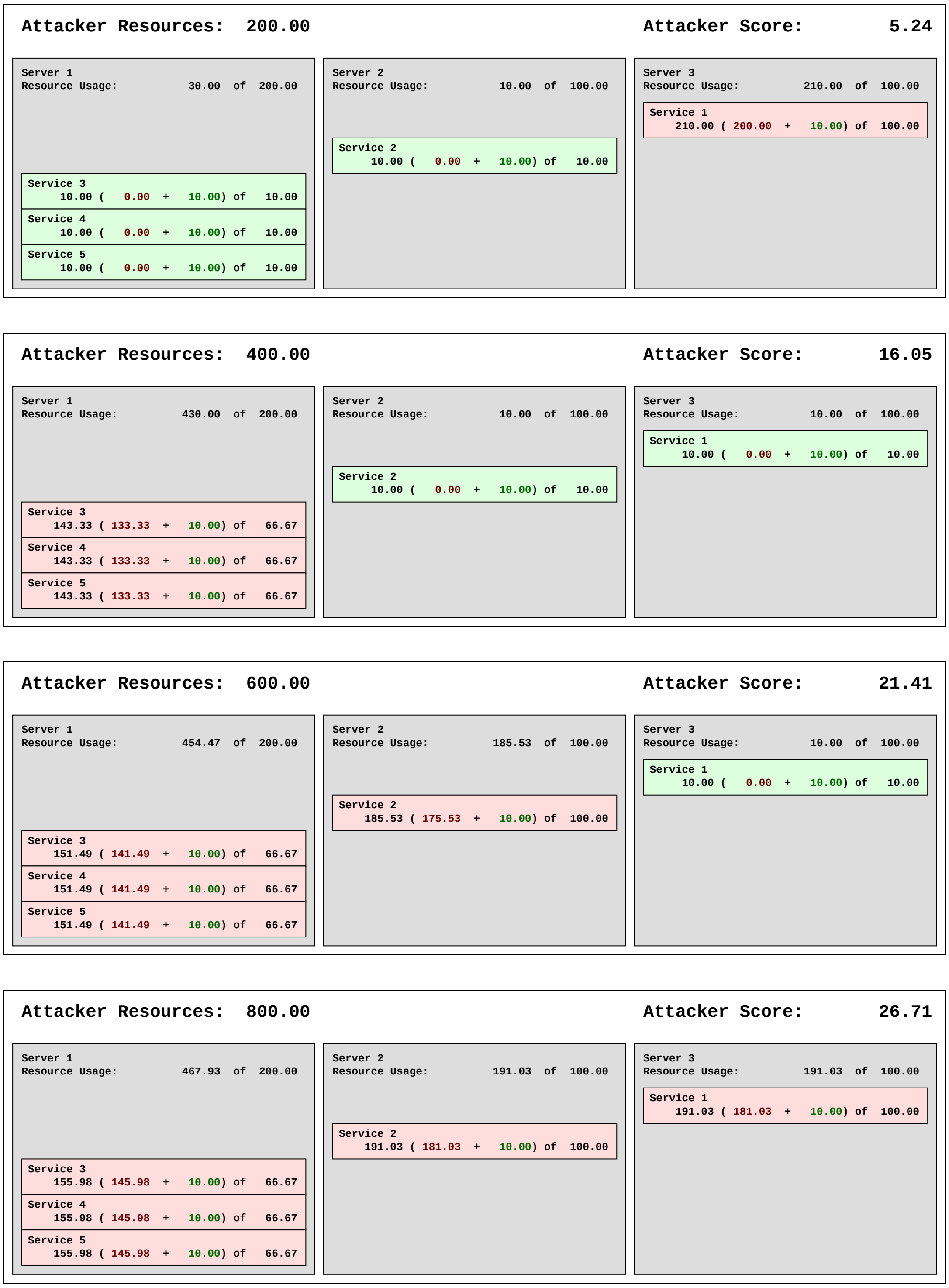
A single Flones 2 simulation run takes two inputs: A scenario (nodes, links, services, attacker, good client) and an attack strategy.

$$simulation(scenario, attack\_strategy) = attacker\_score$$

The attack strategy is the attacker's resource distribution among the potential targets, so analogous to Section 7.2.1, it can be written as a vector $(r_{a,1}, \ldots, r_{a,n})$ with n being the number of potential target services. The attacker's resources are limited, so $\sum_{i=1}^{n}(r_{a,i}) = R$. The output is again a score value based on the good client's packet loss.

While keeping the scenario constant, we run an optimization loop which searches for the best possible attack strategy.

$$\max_{attack\_strategy} simulation(scenario, attack\_strategy)$$



```
Attacker Resources:  200.00                          Attacker Score:        5.24

Server 1                             Server 2                             Server 3
Resource Usage:     30.00 of 200.00  Resource Usage:     10.00 of 100.00  Resource Usage:    210.00 of 100.00
                                                                          Service 1
                                                                            210.00 ( 200.00 +  10.00) of 100.00
                                     Service 2
                                       10.00 (   0.00 +  10.00) of  10.00
Service 3
  10.00 (   0.00 +  10.00) of  10.00
Service 4
  10.00 (   0.00 +  10.00) of  10.00
Service 5
  10.00 (   0.00 +  10.00) of  10.00
```

```
Attacker Resources:  400.00                          Attacker Score:       16.05

Server 1                             Server 2                             Server 3
Resource Usage:    430.00 of 200.00  Resource Usage:     10.00 of 100.00  Resource Usage:     10.00 of 100.00
                                                                          Service 1
                                                                             10.00 (   0.00 +  10.00) of  10.00
                                     Service 2
                                       10.00 (   0.00 +  10.00) of  10.00
Service 3
 143.33 ( 133.33 +  10.00) of  66.67
Service 4
 143.33 ( 133.33 +  10.00) of  66.67
Service 5
 143.33 ( 133.33 +  10.00) of  66.67
```

```
Attacker Resources:  600.00                          Attacker Score:       21.41

Server 1                             Server 2                             Server 3
Resource Usage:    454.47 of 200.00  Resource Usage:    185.53 of 100.00  Resource Usage:     10.00 of 100.00
                                                                          Service 1
                                                                             10.00 (   0.00 +  10.00) of  10.00
                                     Service 2
                                      185.53 ( 175.53 +  10.00) of 100.00
Service 3
 151.49 ( 141.49 +  10.00) of  66.67
Service 4
 151.49 ( 141.49 +  10.00) of  66.67
Service 5
 151.49 ( 141.49 +  10.00) of  66.67
```

```
Attacker Resources:  800.00                          Attacker Score:       26.71

Server 1                             Server 2                             Server 3
Resource Usage:    467.93 of 200.00  Resource Usage:    191.03 of 100.00  Resource Usage:    191.03 of 100.00
                                                                          Service 1
                                                                            191.03 ( 181.03 +  10.00) of 100.00
                                     Service 2
                                      191.03 ( 181.03 +  10.00) of 100.00
Service 3
 155.98 ( 145.98 +  10.00) of  66.67
Service 4
 155.98 ( 145.98 +  10.00) of  66.67
Service 5
 155.98 ( 145.98 +  10.00) of  66.67
```

Figure 7.5: Optimal attacker and defender strategies, Server 1 is twice as strong as Server 2 and Server 3. The attacker's strength is increased from top to bottom.

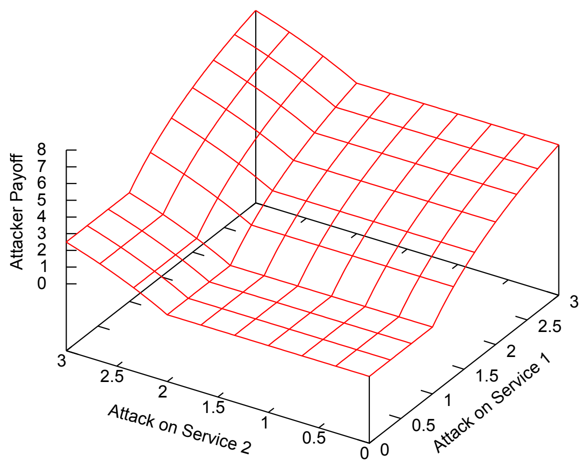


Figure 7.6: DoS optimization as a multidimensional problem: The attacker's total payoff is the sum of the attacks on individual services. The lower part of the graph shows the area, where no service is overloaded, therefore the payoff is zero. The optimizer works on such graphs with usually more dimensions and a constraint on the attacker's resources.

For the choice of optimization algorithms, we have to look at the behavior of our services when the load is increased. A single service corresponds to one dimension of our problem space. Figure 7.6 is a two-dimensional variant of Figure 7.1 to illustrate this.

A service will not loose any packets as long as it is not overloaded. This is a problem when using traditional optimization algorithms like the conjugate gradient method. If the starting point of the optimization is in the area without packet loss, the optimization algorithm will not see an improvement in any direction and therefore stop immediately.

When the load on a service is increased further, the good client's packet loss will show the same concave behavior as in Figure 7.1. However it should be noted that also different loss functions may occur in a Flones 2 scenario. For example a load balancer that is only active if the load on a service exceeds a certain level will lead to discontinuities in the packet loss function.

Concluding we assume that the function to be optimized is piecewise differentiable, which is the case for all scenarios in this chapter. The function may contain a limited number of discontinuities in each dimension, and in each dimension there is a flat area close to zero.

Practical experiments have shown that a combination of optimization algorithms usually yields the best result. First we test many points that lie distributed in the definition space, by using a random search, an evolutionary algorithm or by testing all corners of the definition space. After that a downhill simplex quickly converges to the right point. The optimization is based on standard algorithms from the libraries SciPy[1] and Pyevolve[2].

7.5 Simulation Results

In this section we investigate two scenarios. The first one is shown in Figure 7.7 and based on the DoS attack against Microsoft which was already described in Chapter 3.

[1]`http://www.scipy.org/`

[2]`http://pyevolve.sourceforge.net/`

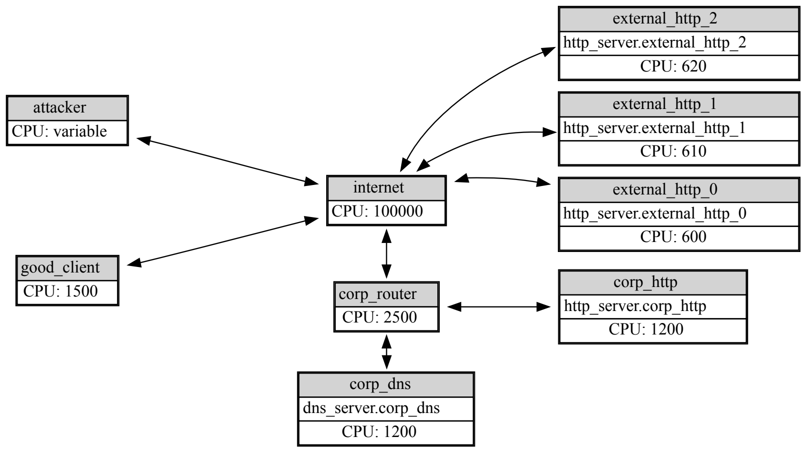
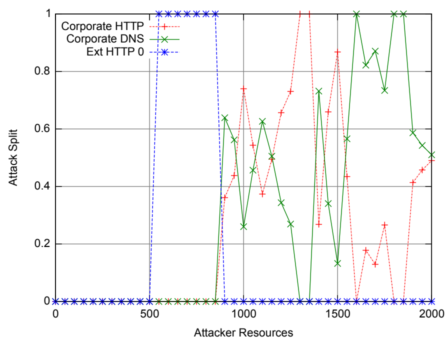


Figure 7.7: MS-DoS scenario.

Here we have four HTTP servers, one of which is located in the company's internal network while the others are directly connected to the Internet. Additionally we have a DNS server which is also located in the company's internal network. The router "corp_router" which connects the internal network to the Internet is the bottleneck of the scenario.

The good client is the aggregation of all users of the HTTP service. Simplifying the real-world DNS protocol, we state that he always has to send a query to the DNS server before being able to send a request to one of the HTTP servers (this is a "client-side direct dependency" combined with a "router dependency" in the categorization of Section 3.1). The single point of failure of the network is the DNS server, while the corporate router is the bottleneck on the way there. The attacker can choose to send packets to each of the HTTP servers or to the DNS server, he is not required to make a DNS query before each HTTP request.



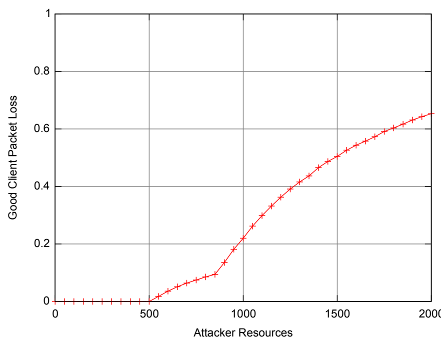


Figure 7.8: Attacker strategy in the MS-DoS scenario.

Figure 7.9: Attack success in the MS-DoS scenario.

We simulated this scenario with different amounts of attacker resources. Each time, the attacker strategy was optimized to find the best possible attack on the network. Figure 7.8 shows the attacker's resource distribution, Figure 7.9 the damage caused by the attacker.

We find that the attacker needs a bit more than 500 resource units per simulation round to cause damage. The weakest spot is one of the external HTTP servers with a capacity of 600 resource units and which is busy for 100 resource units serving

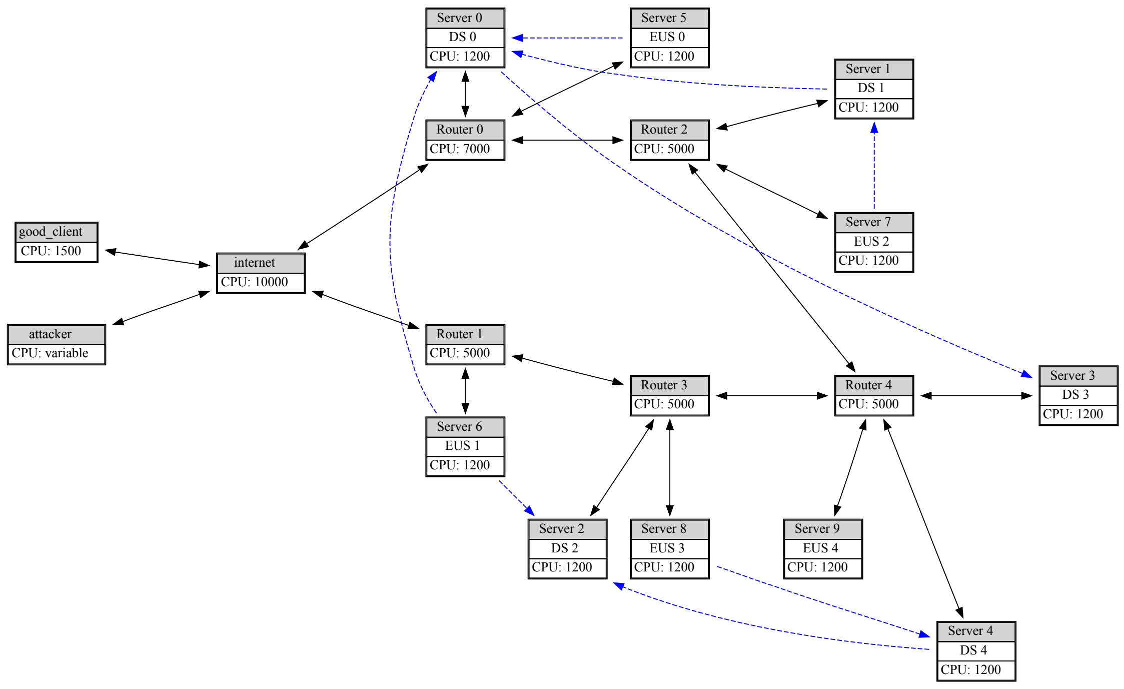


Figure 7.10: Random scenario.

requests from the good client. Starting at 900 units of attack capacity, the attacker switches to attacking services behind the bottleneck router. At this point, the router's load per round consists of 400 DNS queries from the good client, 100 HTTP requests from the good client, 900 requests from the attacker and the corresponding replies for all of these requests, so the router would need 2800 resource units to cope with the load. Causing packet loss at the router would have been possible sooner, but the attack on the external HTTP server still had the higher payoff. When attacking the bottleneck, the attacker is indifferent between the DNS service and the internal HTTP service. Packets to both targets flow through the weak router and therefore have the same effect while at the same time the bottleneck makes it impossible to overload either service.

The second scenario shown in Figure 7.10 was created by a random topology and service dependency generator. The blue dashed arrows are the service dependencies. For example when "End-User Service 0" (EUS 0) receives a request, it has to ask for some information at "Dependency Service 0" (DS 0) which again has to ask back at DS 3 ("server-side direct dependencies" according to Section 3.1).

The results of simulating the random scenario are shown in Figures 7.11 and 7.12. Additionally Figures 7.13 and 7.14 show the optimal attack strategies of attackers with 1500 and 1550 resource units per round against the random scenario. These two graphics were generated by Flones' Graphviz Tracer implemented for [45]. This type of graphical output greatly helps in identifying the actual weaknesses.

The attacker needs 500 resource units to initially cause damage. With this amount of resources he targets EUS 2, however the weak spot responsible for the packet loss is Router 2 which becomes overloaded with EUS 2 messages, DS 1 and DS 0 messages and traffic from the good client. At 1550 resource units, the attacker starts distributing his resources to also attack EUS 0. This causes a massive increase

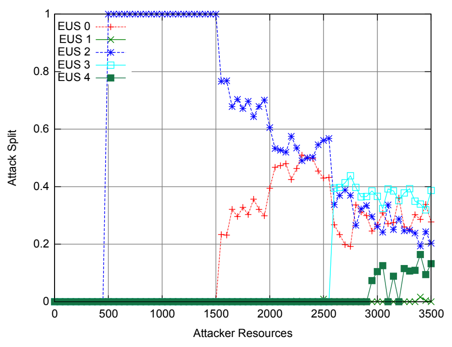


Figure 7.11: Attacker strategy in the random scenario.

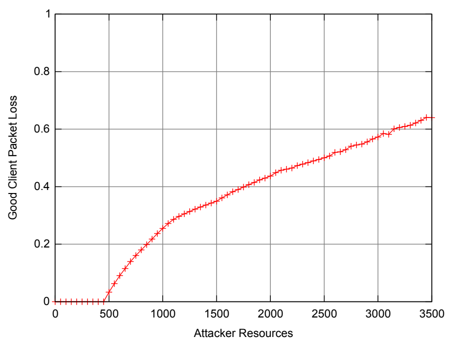


Figure 7.12: Attack success in the random scenario.

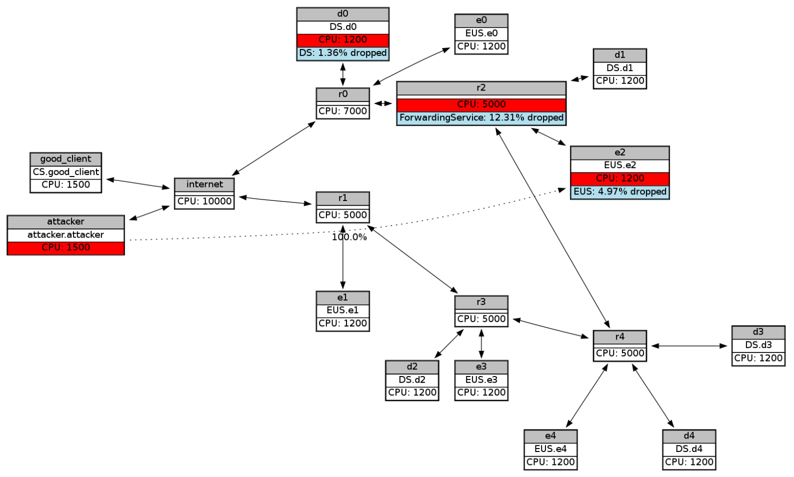


Figure 7.13: Optimal attack against the random scenario for an attacker with 1500 resource units per round. Only e2 is attacked.

in the load of DS 0 which becomes the new main target. At 2600 resources units the attacker opens up a new front by targeting EUS 3. Again the main load is not created at the targeted service, but in Routers 3 and 4, amplified by of EUS 3's dependencies. At 2950 resources the attacker finally adds EUS 4 to its list of targets.

As we see in this scenario, the attacker actually increases the number of targets when being supplied with more resources. This way he can maintain an almost linear increase in his payoff, despite the concave nature of the individual payoff functions.

7.6 Hardening the Network: The Defender Side

In this section we add an outer optimization loop to the simulation to make the defender improve his network. In game-theoretic terms we calculate a minimax solution:

$$\min_{DefenderStrategy} \max_{AttackerStrategy} AttackerScore$$

or equivalently

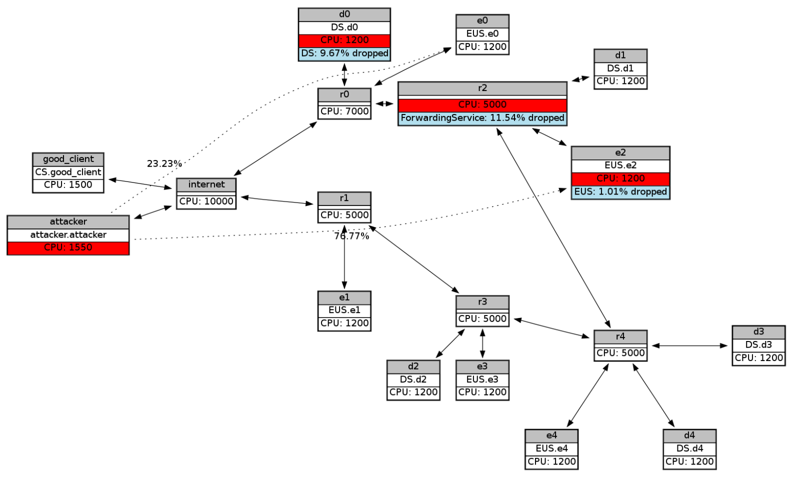


Figure 7.14: Optimal attack against the random scenario for an attacker with 1550 resource units per round. The attacker adds e0 to the set of attacked hosts.

$$\max_{DefenderStrategy} \min_{AttackerStrategy} DefenderScore,$$

where the attacker's score is again the good client's packet loss. On the other hand the defender's score is based on the successfully served good client requests. We will use the defender's score as the main metric in this section.

John von Neumann's minimax theorem only guarantees the existence of this solution for mixed strategies. In our cases, solutions exist but are generally not unique — as we saw in the previous Section 7.5 there are trivial examples of situations where the attacker is indifferent between attacking two services.

While we continue using standard conjugate gradient and downhill simplex optimizers for the attacker optimization, we completely switch to evolutionary algorithms for the defender. This is because the inner optimizers usually do not hit the optimum perfectly but show random fluctuations. A Flones 2 simulation run is a complex calculation based on 64-bit floating point numbers. The limited floating point accuracy introduces errors that accumulate during a simulation run. They become significant when trying to calculate gradients on the results of multiple simulation runs. One can say that the outer optimizer sees a "rough" landscape that is has to run on.

Evolutionary algorithms mainly test random points of the whole definition space and then narrow down the search on areas where good solutions were found previously — this procedure makes them more stable in our situation. However as will be shown below, they also usually do not find an optimum.

The combination of inner and outer optimizations also takes relatively long to analyze. Simulation runtimes for the two scenarios investigated below were in the order of 4-6 hours when both optimizations were used. A pure attacker optimization usually takes less than a minute.

The MS-DoS scenario already presented in Section 7.5 was tested with the new defender. Only the resource allocation among the different servers was optimized. One aspect that might be interesting for future work is the investigation of changes to the topology — the defender would be allowed to connect his nodes differently. However this would make the definition space of the problem even larger, therefore it is not clear, how topology-changes for the defense can be simulated effectively.

In the currently investigated MS-DoS scenario as shown in Figure 7.7, there are six nodes under the control of the defender. The defender has 70 000 resource units to distribute among them, the attacker can distribute 20 000 resource units. As before, the attacker needs one resource unit to send one request while the defender requires one resource unit to answer it.

The router at the entry of the corporate network also requires one resource unit to forward a message. This router is not attacked directly, but traffic to the DNS and corp_http servers generates load on this router. Again, the single point of failure of this network is the DNS server, while the corporate router is the bottleneck on the way there.

Node	Attacked	Defended
Router		X
DNS	X	X
corp_http	X	X
http0	X	X
http1	X	X
http2	X	X

During the optimization, 642 different defender strategies were tested by the evolutionary algorithm. Table 7.1 shows the 10 best strategies that it encountered while optimizing, each one with the corresponding optimized attacker strategy. We see that in all defense strategies, the defender assigns the highest amount of resources to the router and almost always the second-highest to the DNS server. The optimizer correctly detected the single points of failure on this network.

Two different strategies achieved the highest defender score of 3074. Flones 2 graphs of these simulation runs can be seen in Figures 7.15 and 7.16. These graphs were obtained by adding a tracing to the simulator, which produces a graphical output.

In both graphs the defender assigns very small amounts of resources to two of the external HTTP servers, while the 3^{rd} one gets more computing power. These strategies are clearly not optimal as the uneven distribution wastes resources on server http0.

Figure 7.17 shows a manually chosen defense strategy for the MS-DoS network with a score of 4690 — a significant improvement compared to the evolutionary optimization.

The second scenario, called cascading scenario in the following, is a smaller version of the random scenario from Section 7.4. It is shown in Figure 7.18.

Here we have three end-user services (EUS) that depend on three back end services (dependency services, DS). The back end services are not directly accessible to the attacker. The defender distributes resources among all six services. The routers in this scenario have plenty of resources, so they are not bottlenecks.

Node	Attacked	Defended
EUS 0	X	X
DS 0		X
EUS 1	X	X
DS 1		X
EUS 2	X	X
DS 2		X

Score	Defender Strategy (%)						Attacker Strategy (%)				
(Defender)	Router	DNS	corp_http	http0	http1	http2	DNS	corp_http	http0	http1	http2
3074	27.8	27.8	22.2	11.1	5.6	5.6	23.7	28.2	0.0	24.0	24.0
3074	27.8	22.2	16.7	22.2	5.6	5.6	28.2	23.7	0.0	24.0	24.0
2778	40.0	20.0	0.0	10.0	20.0	10.0	0.0	100.0	0.0	0.0	0.0
2747	26.3	26.3	21.1	15.8	5.3	5.3	22.9	29.7	0.0	23.7	23.7
2484	27.8	22.2	5.6	16.7	5.6	22.2	30.0	44.8	0.0	25.2	0.0
2483	31.2	25.0	6.2	18.8	12.5	6.2	35.0	65.0	0.0	0.0	0.0
2456	27.8	16.7	16.7	5.6	11.1	22.2	21.4	54.0	24.7	0.0	0.0
2355	38.5	38.5	7.7	7.7	7.7	0.0	25.4	74.6	0.0	0.0	0.0
2330	55.6	11.1	11.1	11.1	0.0	11.1	0.0	100.0	0.0	0.0	0.0
2263	29.4	23.5	23.5	5.9	11.8	5.9	41.8	58.2	0.0	0.0	0.0

Table 7.1: Best optimization results of MS-DoS scenario.

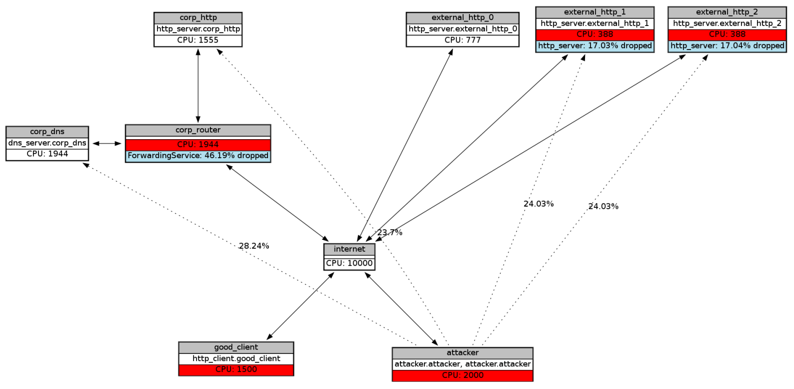


Figure 7.15: Best defense result 1 for the MS-DoS scenario.

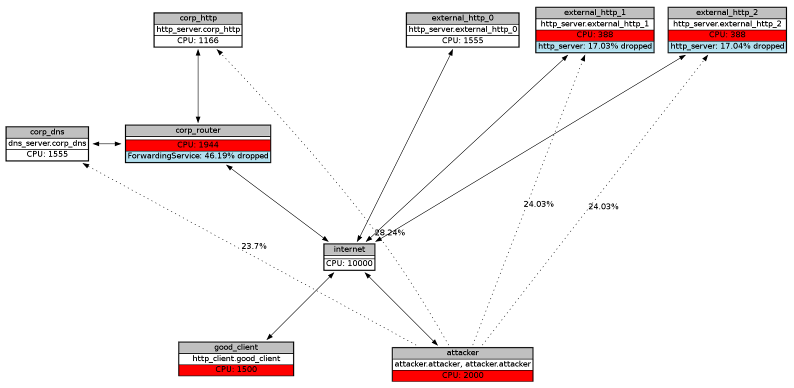


Figure 7.16: Best defense result 2 for the MS-DoS scenario.

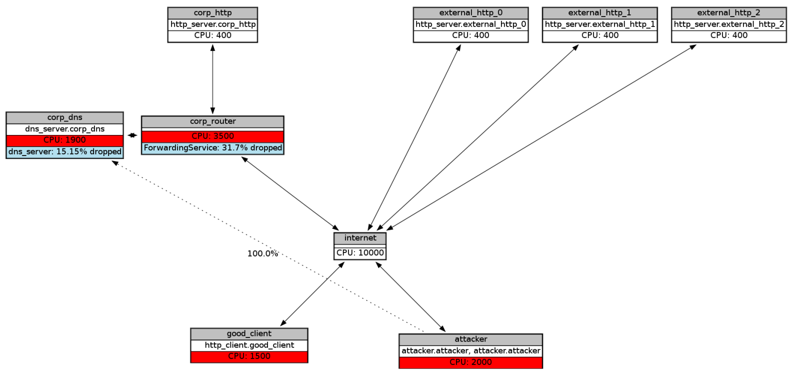


Figure 7.17: Best manual defense for the MS-DoS scenario.

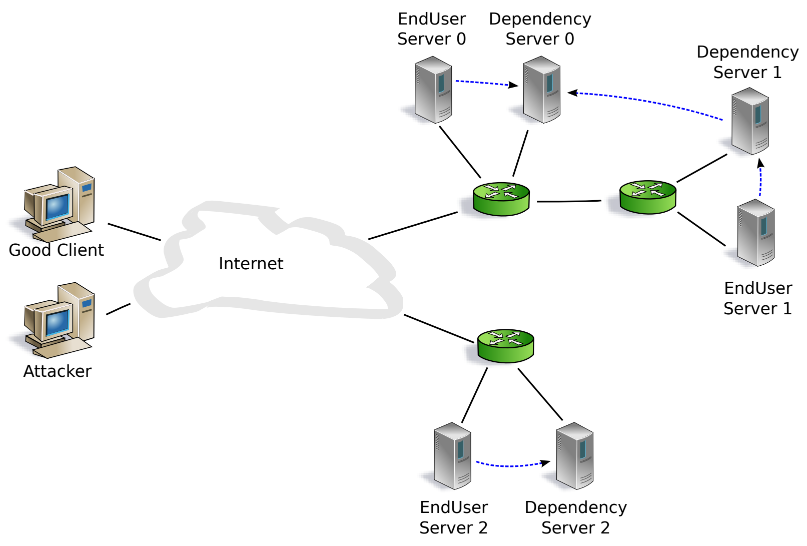


Figure 7.18: The cascading scenario.

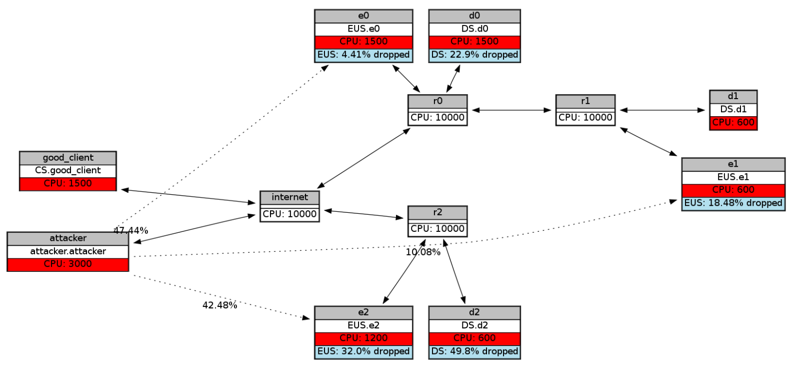


Figure 7.19: Best defense result for the cascading scenario.

The defender can distribute 60 000 resource units while the attacker has 30 000 resource units available. We see that DS 0 is a potential weak spot in the network, as both EUS 0 and EUS 1 depend on it.

Table 7.2 shows the 10 best optimization results of the 680 configurations that the evolutionary algorithm tested. The best result is also shown as a Flones 2 graph in Figure 7.19. We see that the weak spot at DS 0 is recognized, this server is supplied with a lot of resources. However, even the best distribution is clearly suboptimal, as it assigns 20% of the total resources to EUS 2, but only 10% to DS 2. As EUS 2 depends on DS 2 and as the resources scale equally, EUS 2 will never be able to serve as many requests as its resources permit.

Figure 7.20 shows a manually selected defense strategy, where this flaw has been fixed. The spare resources from EUS 2 have been equally distributed among all servers. This raises the score by a small amount from 13572 to 13654.

Concluding we can see that the optimization produces good but not optimal results. With the cascading scenario, the manual improvement has not been as impressive as with the MS-DoS scenario. However, in the cascading scenario, manual improvements are not that obvious either.

Score	Defender Strategy (%)						Attacker Strategy (%)		
(Defender)	DS0	EUS0	DS1	EUS1	DS2	EUS2	EUS0	EUS1	EUS2
13572	25.0	25.0	10.0	10.0	10.0	20.0	47.4	10.1	42.5
13438	22.7	22.7	9.1	9.1	13.6	22.7	43.2	7.2	49.7
13156	25.0	25.0	0.0	8.3	16.7	25.0	50.2	0.0	49.8
13140	23.8	23.8	9.5	9.5	9.5	23.8	45.7	8.8	45.4
12989	20.0	20.0	0.0	6.7	26.7	26.7	49.3	0.0	50.7
12805	27.8	16.7	11.1	11.1	16.7	16.7	36.2	14.0	49.8
12759	21.1	26.3	10.5	10.5	10.5	21.1	45.5	9.0	45.5
12726	23.1	23.1	0.0	7.7	15.4	30.8	50.5	0.0	49.5
12705	23.8	23.8	9.5	4.8	14.3	23.8	50.4	0.0	49.6
12690	19.0	23.8	9.5	9.5	14.3	23.8	43.2	7.1	49.7

Table 7.2: Best optimization results of the cascading scenario.

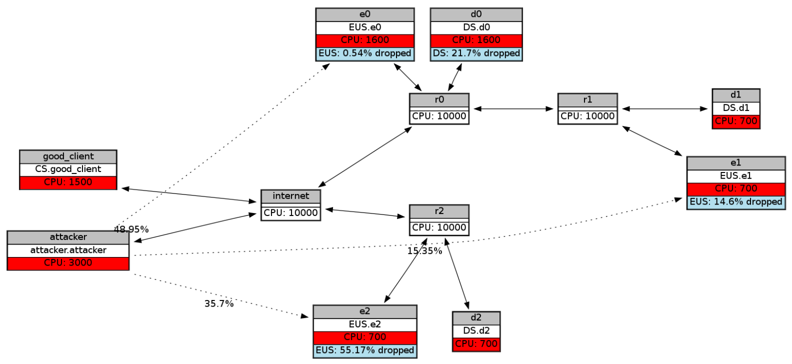


Figure 7.20: Manual better defense for the cascading scenario.

The reason for the sub-optimality is that the two nested optimizations take very long to complete. The evolutionary algorithm would find better defense strategies if it had more time, however simulation durations were already extreme for the solutions presented here and the very small simulated networks. Further research, especially in the field of optimization is required to develop a solution that works on larger networks.

7.7 Practical Considerations

To perform the attacker analysis as well as the defender analysis that we presented here, a model of the investigated network is required. This section is intended to provide hints on how to build a system which automates this task.

As already discussed in Section 3.2, network inventory is a task of network administrators. Ideally the administrator of a company network should have a plan of the network topology, hardware devices and also the services and their dependencies. Such a plan may be available in the form of text files and drawings, however for a large network the use of a management software like HP Open View[26] is more desirable.

Section 3.2 also discussed that there is a lot of related work regarding dependency discovery of networked services.

One more necessary step is discovery of available resources and resource consumption. We need information about the resource usage of each request type and in case of multiple services on a single machine, the behavior of the server when faced with a mixed load. If the network is sufficiently outfitted with monitoring agents — ideally on every server — this can be discovered automatically by learning from measurement values during normal operations. If this is not the case, active probing may be required.

When all data has been collected, a model of the network for Flones can be generated in the form of a Python program. The process of defining networks and services in Flones was inspired by the TCL-based simulation scripts of NS2. The actual Flones simulation core is included as a module which is called with the network topology as a parameter.

As shown above, a static analysis of a network for discovering weaknesses works very well. Automatically finding an optimal defense is computationally expensive, so in the worst case the network administrator would manually build an improved topology based on the acquired knowledge of the old infrastructure's weaknesses and then test the new model with Flones before making changes to the real network.

7.8 Conclusions

In this chapter we have shown that DoS attacks can be modeled as a sequential variant of the well-known Colonel Blotto Game with a payoff-function based on a good client's packet loss. Dependencies between services have the potential of creating bottlenecks which allow a DoS attacker to cause huge amounts of damage with little effort. Such weak spots in a network can be found when optimizing an attacker's strategy using a network simulator. If no bottleneck can be exploited, the attacker will concentrate his attack on few targets if his possibilities are limited. A stronger attacker can afford to attack a larger number of targets at once.

We have developed a method which can identify weak spots in network topologies that might be exploited by a DoS attack. This method is designed to be used by the owner of the attacked network — the defender — to investigate its own infrastructure and fix weaknesses there.

An automatic improvement of the network topology is currently only feasible for small network topologies, as it is computationally expensive.

Key findings in Chapter 7:

- **We use the Good Client's packet loss as a metric for an attack's success. When increasing the attack traffic to a service, this metric follows a concave curve: When the service is already overloaded, further increasing the strength of the attack gives the attacker less and less benefit.**
- **An attacker with limited resources will concentrate his attack to a single target. The stronger the attacker is, the more targets he wants to attack.**
- **If multiple services have to be defended and all of them are equally valuable, the defender should split the defensive resources equally.**
- **In the case that m services run on n physical servers with $m>n$, the attacker will first choose the servers to attack, depending on his resources. Then he will attack all services on the chosen servers.**
- **Weaknesses in networks can be found by optimizing the attacker's strategy. However, adding an outer optimization for improving the network leads to very long simulation times.**

7.9 Outlook

"Flash Crowds" of visitors on a web site can create similar overloads as DoS attacks. The methods investigated here are suitable for improving the worst-case flash crowd behavior of a network.

If the goal of an analysis is protection against a flash crowd rather than a DoS attack, one key assumption changes. For DoS attacks we assumed that the attacker will attack the set of services that causes the maximum damage. For a flash crowd, the defender will be able to give hints on which services might potentially be affected and which ones are completely uninteresting for normal users. Therefore a flash-crowd analysis will rather focus on the public web servers than on some internal website which requires a user account.

To achieve this, the recommended procedure is to model the complete network including the uninteresting services and to assign a good client usage to each of them. However, the options of the attacker should be limited to services where a flash crowd appears possible. This makes sure that dependencies which receive load from the public servers as well as from internal services are calculated correctly.

The methods developed in this chapter also appear suitable for resource dimensioning of *Web Services*. One key property of web services that represent business transactions is the amount of service dependencies. Flones would be able to simulate such networks and find resource bottlenecks.

7.10 Acknowledgements

The author would like to thank Heiko Niedermayer for discussing the topics in this chapter. Alexander Gitter wrote parts of the Flones 2 simulations presented here and developed the components that generated Figures 7.13 and 7.14. Bernd Schultze worked on the defender optimization.

8. Attacks against a Dynamic Defender

As was shown in Section 4.2.5, timing is a huge issue in Colonel Blotto Games. In the Classical Blotto Game, both players act simultaneously — a fact which leads to the complicated properties of the game. If the two players act sequentially, the second player has a major advantage as he can observe the first player's decision and react accordingly. When the resources of both players are almost equal, the second player will always win.

The DoS game introduced in Chapter 7 is basically a sequential Colonel Blotto game. First the defender builds his network, a decision which can not be changed easily afterwards. Changes to the network topology and servers have to be conducted manually, are costly and possibly make the services unavailable during the time of the change. Therefore the defender is inflexible, during the attack he has to stick with his prior decision. The attacker on the other hand has time to investigate the network's properties, i.e. by trying different attacks and observing the results. He is flexible to change his choice at any time.

In modern data centers, services often run on virtualized hardware. Virtualization abstracts from the underlying hardware and allows for more efficient resource usage, as multiple virtual machines can share a physical machine. It also can be used for security purposes, as it allows to isolate services.

The feature which makes virtualization interesting in our context is live migration. Virtual machines can be moved from one physical machine to another, the actual downtime during this process is very short and barely noticeable.

Therefore virtualization can be used to change the defender's service allocation quickly during an ongoing attack.

We will discuss related work in Section 8.1. Section 8.2 is about the performance of migrating web servers under high-load conditions with XEN virtualization. The scenario for this chapter is outlined in Section 8.3, Section 8.4 gives strategies for attacker and defender. Simulation results are presented in Section 8.5 while Section 8.6 concludes this chapter.

8.1 Related Work

Virtual machines are widely used in today's data centers, especially in the context of cloud computing. A physical server contains several virtual servers, each of which runs its own operating system and set of services. A scheduler cares for the distribution of physical resources between the virtual machines. Common virtualization solutions are KVM[1], XEN[2], VirtualBox[3] and VMWare[4].

Virtualization solutions allow for moving virtual machines between physical machines. During a "cold migration", the virtual machine is turned off. "Live migrations" move a virtual machine at runtime. This is done by copying the state of the running virtual machine to the new physical machine while the machine is still running on the source host. Since the machine's state changes during the process, the differences are copied in a second step. If the states on both sides are sufficiently similar, the machine is turned off on the source host and finally started on the destination host. Therefore there is still a short downtime of usually less than a second.

One option of using virtualization for improving the defender's chances during an ongoing denial-of-service attack is cloud computing. Virtual machines on Amazon EC2 can be started within minutes, theoretically it is possible to start thousands of servers. However, they are charged by usage which can be very expensive especially when a lot of data is transmitted (0.08 US-$ - 0.19 US-$ per Gigabyte at the time of this writing[5], which translates to 36 US-$ - 85.50 US-$ per "gigabit-hour"). If the attacked service is valuable enough, renting resources in the cloud is a valid option for defense.

A proof-of-concept has shown, that also attackers can rent resources in the cloud for denial-of-service attacks[6].

DoS defenses using virtualization have also been suggested by the authors of [46]. We developed this idea independently from them because of game-theoretic considerations. They show in an experiment that migration of a virtual machine using XEN can improve load conditions during a DoS attack. Experiments with XEN under DoS conditions have also been conducted for this thesis, see Section 8.2.

The focus of this thesis differs from that of previous work, as our main subject of research is the strategic behavior of attacker and defender, which was not considered before.

There has been related work, in the field of load balancing server farms. Algorithms were developed to handle legitimate load, for example in [47] and [48]. These algorithms distribute services among physical servers based on the observed load in the past. We will see in the following sections that an attacker might be able to exploit this property.

[1] `http://www.linux-kvm.org/`
[2] `http://www.xen.org/`
[3] `http://www.virtualbox.org/`
[4] `http://www.vmware.com/`
[5] `http://aws.amazon.com/de/ec2/pricing/`
[6] `http://www.h-online.com/security/news/item/Thunder-from-the-cloud-1051917.html`

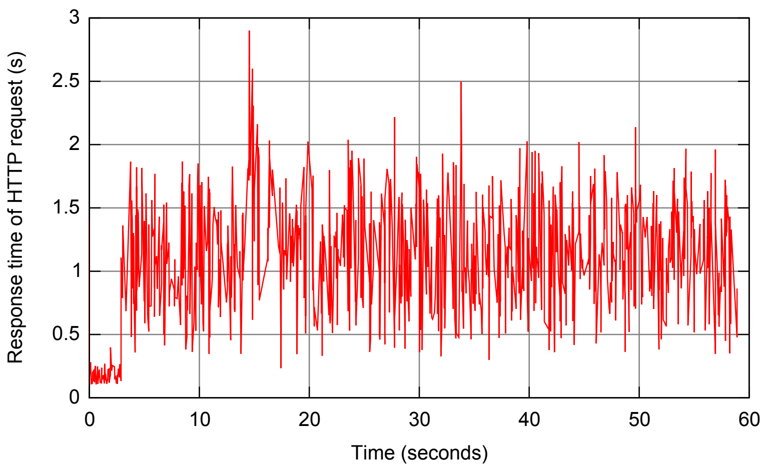


Figure 8.1: Migration of a small virtual machine.

8.2 Motivating Experiments

Various experiments on live migration of loaded services have been conducted during the research for this thesis. This section presents results of a testbed-experiment that was conducted using the XEN hypervisor[7].

The setup consisted of a client and two physical servers, each one equipped with a Core 2 Duo processor (with support for hardware virtualization) and 4GB of RAM. As services, HTTP servers were placed in “virtual appliances” — ramdisk-based virtual machines. The virtual servers were tested with two different RAM configurations, the small variant had 64 MB of RAM, while the large one used 1280 MB.

The client was an Apache Benchmark. This benchmarking tool can be configured to send a number of requests (i.e. 10) simultaneously. A new request is sent as soon as a reply for one of the old requests arrives, therefore the number of concurrently active requests is kept constant. In this scenario, no requests remain unanswered.

Each HTTP request triggered a CPU-intensive CGI script that took 0.1 seconds to run, therefore delaying the response. The expectation is that one of the dual-core servers can serve 20 requests of this type per second.

Figure 8.1 shows the response-times of a virtualized web server with 64 MB of RAM during a migration. The machine is constantly loaded with 20 concurrent requests during the process and migrated to a new server at second 15. One can see a peak in the graph at this time, which is caused by background load when the virtual machine’s state is transferred to the new server. As this is a live migration, the actual downtime of the virtual machine is less than one second.

The total cost of the migration measured in fewer processed requests is moderate. Without migration the server could typically process 1034 requests within 60 seconds, with the migration this number dropped to 1018. These results were repeatable, so the costs of a migration can be assumed to be equivalent to the processing of 16 (CPU-intensive) HTTP requests.

In Figure 8.2, the same migration experiment is performed on a virtual machine with 1280 MB RAM. In this experiment, the response times increased for a period of

[7]`http://www.xen.org/`

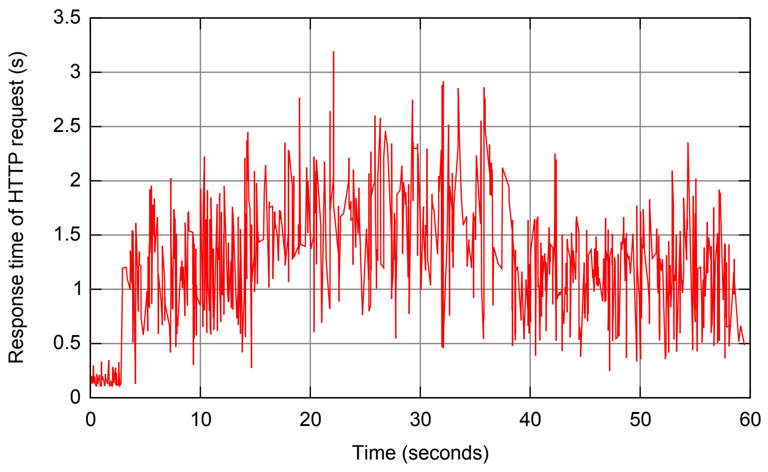


Figure 8.2: Migration of a large virtual machine.

25 seconds. Again the service remained responsive during the migration, the actual downtime during the migration was again less than a second.

In this experiment the number of processed requests dropped to 909, so the migration costs can be assumed to be equivalent to 125 HTTP-requests.

Finally Figure 8.3 shows a DoS mitigation experiment. Here two virtual servers run on a single physical server. Each virtual server is loaded with 10 concurrent requests. A DoS detection mechanism moves *virtual server 2* to an unloaded physical server as soon as the average load of the physical server in a 10-second window is above 90%. The 64 MB virtual appliances were used for this experiment.

The migration is triggered after 13 seconds. After that the response times of the web servers are reduced dramatically. Again the migrated virtual machine is unavailable for approximately one second.

The situation was compared to the same scenario without defense. If both virtual machines remain on the same server, they are able to serve 508 and 509 requests within 60 seconds. If the defense is triggered as defined above, the total amount of processed requests increases to 923 for virtual server 1 and 910 for the (migrated) virtual server 2.

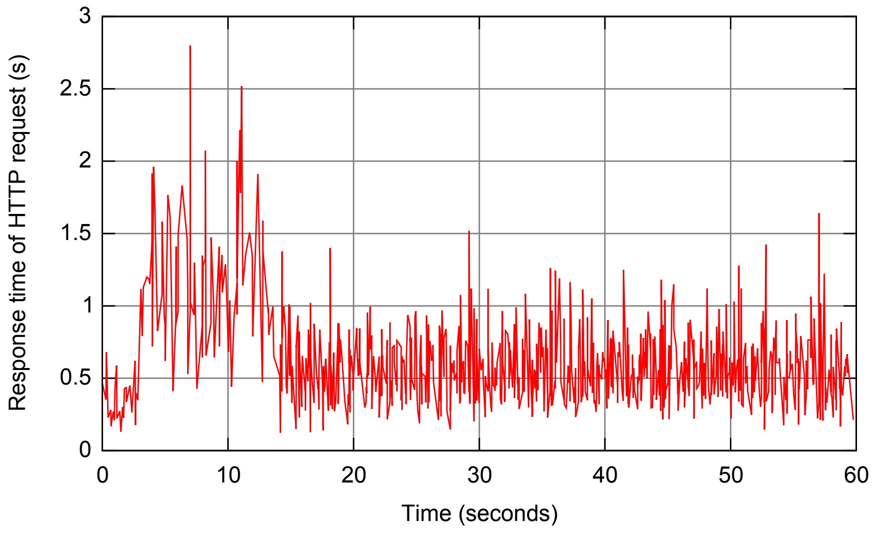
Concluding we can say that live migration comes at moderate costs. Migration as a DoS defense is only useful, if the benefit of a migration is higher.

8.3 Scenario and Assumptions

The added flexibility of virtualization has its price. There is generally some overhead because of virtualization, however, as today's virtualization solutions are optimized for running servers and as virtualization is also used because of other advantages (e.g. cost reductions because of better server utilization, enhanced security due to the isolation of virtual machines), we assume those static costs to be negligible. However, as shown in the previous section, live migration requires the transfer of hundreds of megabytes of state information from one machine to another which makes migrations costly.

Therefore it is also clear that an attacker who knows that virtualization is used as a DoS defense would want to trick his opponent into moving machines unnecessarily

(a) Virtual server 1



(b) Virtual server 2

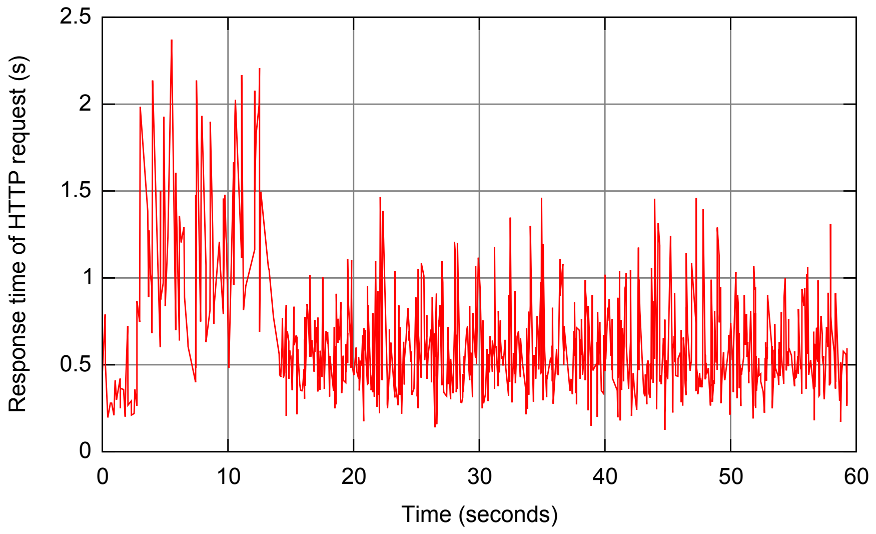


Figure 8.3: Load-balancing during a DoS attack. Initially both web servers run on the same physical machine. When the attack is detected, guest 2 is moved to an unloaded server.

(this would fall under “Additional vulnerabilities created” in the taxonomy for DoS defense evaluation in Section 9).

The following sections discuss assumptions regarding the scenario.

8.3.1 Information

In reality, moving virtual machines (usually within a single data center) without losing network connectivity is done on Layer 2. There is a “virtual bridge” inside each of the physical servers which makes Ethernet connectivity to the outside network available inside the virtual machines. Therefore virtual machines can be moved without losing their (possibly globally routeable) IP address. An alternative setup would consist of a load balancer at the edge of the network which hides the physical locations of the services inside.

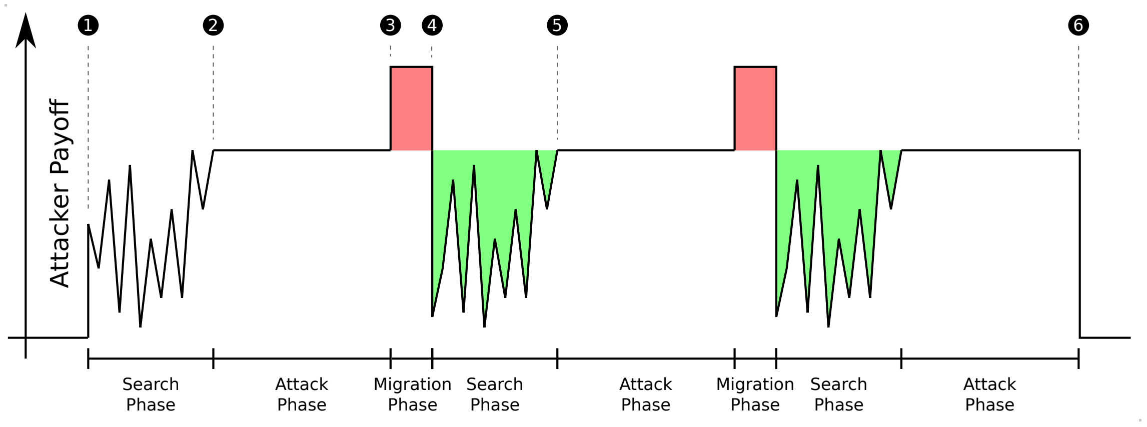


Figure 8.4: Attack Dynamics. The vertical axis shows the attacker's payoff, i.e. lost packets of a good client. The horizontal axis is time.

In both cases we assume that the attacker does not have complete knowledge of the attacked network. In case of a service migration, the attacker will notice the changed service capacity and the migration overhead, so he will know that something has changed. However as changes happen locally within the defender's network, the attacker will not know the new assignment of virtual machines to physical servers.

To get current information, the attacker will have to do probing by running different attacks and observing the results. The attacker is smart and will try to react as well as possible, but the attacker's limited knowledge gives the defender an advantage, at least for a certain time.

8.3.2 Dynamics

The space of possible strategies for attacker and defender is very large. For this thesis we reduce the strategy space by restricting ourselves to defender strategies that follow a certain timing pattern. The defender will decide whether or not to move services at fixed points in time. A strategy in which moving virtual machines can directly be triggered by changes in the attack is potentially more exploitable. In our case the decisions are made in fixed intervals to limit the maximum amount of resources which is spent for moving services.

The above assumptions lead to a behavior of attacker and defender that is shown in Figure 8.4.

At (1) the attack starts. The attacker initially does not know how he can successfully attack the defender's network, so he has to do probing. This means that he tries different attacks and observes the amount of damage done.

The process is similar to the attacker optimization in Chapter 7, the attacker tests different strategies. However, now he knows that he only has a limited amount of time available until the defender will react. Searching for the optimum will probably take too long and the attacker's payoff during search periods is lower than the payoff of constantly sending a reasonable attack pattern. So eventually (2) the attacker chooses an attack which has proven to be good — but which is probably not optimal.

At (3) the defender decides to change his attack. Between (3) and (4) there is an increased resource consumption on the moved services — as discussed above, moving services is costly. The total costs of moving services (in attacker payoff units) is the shape colored red.

At (4) moving the services has ended and the defender is fully operational in his new network state. As the defender reacted to the attacker's actions, the attacker's last attack is probably ineffective now. The attacker now has to start his search for an attack strategy again — and during the search he will have a lower payoff than before. The area marked green is the defender's benefit from moving the services in attacker payoff units.

The whole process will repeat now. The attacker decides on an attack at (5) which remains static until the defender decides to change his network again. The whole process continues to iterate until at some point one of the opponents gives up (6).

From the picture we see that the duration of the defender's cycle and the duration of the attacker's search are important parameters to be investigated. This will be done in later sections of this chapter.

8.3.3 Scheduling

Schedulers of virtual machine monitors try to assign a fair share of resources to each virtual machine. Therefore in the following we will continue with the proportional share scheduler that was introduced in Section 6.3.4 and already used in Chapter 7. Own experiments with XEN in [49] have shown this to be realistic.

8.4 Strategies

This section discusses different strategies for the attacker and defender. We separate them into three categories, simple and clearly sub-optimal strategies, heuristics that we will use for the simulations and advanced strategies might not be realistic in practice, but help to estimate what is theoretically possible.

8.4.1 Simple Strategies

This section describes some very simple attack and defense strategies. These are not useful in real scenarios and are shown as reference points for the evaluation of later improved strategies.

The *static defender* only uses a fixed configuration which is manually assigned at the beginning of the simulation. This will generally be the optimal static distribution according to the considerations in Section 7.3 — the one that minimizes the damage of an attacker who plays optimally.

This strategy can be considered state-of-the-art on the defender's side, as DoS protection by moving services is not used yet. The main goal of this chapter is to show, if an improvement over this best-possible static assignment can be achieved.

The *random attacker* first selects a subset of services to attack. Each possible target service is selected with 30% probability. After that the resources are randomly distributed between the targets by drawing k uniformly distributed random numbers d_i, where k is the number of attacked services.

The distribution $r_1, \ldots, r_k$ of the attacker's total resources R is calculated as $r_i = d_i \cdot \frac{R}{\sum_{i=0,\ldots,k} d_i}$.

This distribution is motivated by the observation that it is rarely optimal to attack all services. Initial tests during the implementation showed that this attacker performs significantly better than an attacker who just calculates a uniform distribution among all services. The choice of the target selection probability will be further elaborated in Section 8.5.

The *static attacker* initially searches for a good attack pattern by trying distributions and evaluating the attack's success. For this he employs the same random selection as the *random attacker*. This search phase only continues for a limited time, after that the *static attacker* constantly sends the best pattern found so far. Once the *static attacker* has entered the static attack phase, he does not monitor the success of the attack, so he continues sending the same pattern even after the defender re-configured his network. This can be considered a state-of-the-art attacker, as he attacks a place where he can do decent damage, but he is not aware of the defender's possibility to adapt.

8.4.2 Heuristic Strategies

The following strategies are improvements on the attacker's and defender's side. They are still easy to implement and expected to be much more effective.

The *bin packing defender* will decide about a new strategy in fixed time intervals. He observes the load on his services caused by attacker and good client. The problem of finding a new assignment of services is a Bin Packing problem (similar to the Knapsack problem), which is known to be NP hard. However, in many cases simple heuristics give good solutions.

In our case, an additional constraint is given by the costs of moving services. Therefore for each candidate assignment of services to servers, a score was calculated, based on the predicted overload using this new configuration and the number of migrations that are necessary to get from the current assignment to the new one. Using this score, a good new assignment was determined using an evolutionary optimization algorithm.

The *heuristic attacker* is an extension of the *static attacker* that continues to monitor the attack's success. Upon noticing that something has changed (which is already the case when the defender starts moving the services), the attacker starts a new random search phase.

Practically, the new random search can be triggered twice in each cycle, at the beginning and at the end of the migration phase. The attacker might find a new good configuration during the defender's migration phase which is no longer valid as soon as the migration has ended. In this case the attacker will notice the change due to the end of the migration phase only after his search phase has ended and immediately start a new search phase.

8.4.3 Advanced Strategies

In this section we discuss strategies that are allowed to use global knowledge.

It is not clear what a real optimal attack strategy would look like. It would surely involve some sophisticated search pattern that quickly gives the attacker a lot of information about the attacked network. An optimal strategy might combine attacking and searching by testing variations of patterns that are already known to yield a good payoff.

The *cheating attacker* estimates the harm that an optimal attacker would do. It simulates search phases which are identical to the heuristic attacker. However, the results of these searches are not used.

After the search phase is over, the cheating attacker gets access to the defender's real service distribution. Using this information, he is able to calculate the optimal attack using an optimization algorithm. In other words this attacker only fakes the search and then continues with the theoretically optimal attack.

Until now, defenders only took the current attack and the costs of moving services as a basis of their decisions. Actually considering the current attack is not very helpful as the attacker can change it immediately after observing a change on the defender's side. With the *heuristic attacker* and *cheating attacker* discussed above the defender has a high risk of choosing a new strategy which is good against the old attack, but very vulnerable against new ones. Therefore the general strength of the new choice should be the main criterion.

Therefore the *anticipating defender* iterates over all possible assignments of services to servers (which is still possible with the 3 servers and 5 services used in the example). For each assignment he calculates a score based on:

- The attacker payoff of the new assignment versus the current attack. This should be a very low value, ideally zero.
- The cost of migrating from the old assignment to the new one.
- The attacker payoff of the new assignment, given that the attacker will chose a new optimal attack. In other words the attacker is expected to behave like the *cheating attacker* discussed above. This payoff is the main criterion which has a weight of 80%.

This defender was used only in few simulation runs as the calculation of the defender's strategy (which must be a best response to the optimal attacker strategy and therefore again involves two nested optimizations) is computationally expensive.

8.5 Results

Figure 8.5 shows the simulation scenario. Default value for the attacker search interval is 60 rounds, for the defender interval 200 rounds.

Figure 8.6 shows a simulation of an *heuristic attacker* against a *bin-packing defender*. On the horizontal axis, the attacker's search interval is increased. It can be seen that it is optimal for the attacker to search only for a very short time interval (here 10 of 200 rounds) to be able to sustain the strongest attack for the remaining time.

If the attacker searches for a little less than 200 rounds, he becomes even worse than a random attacker (which would have a score of 0.08 in this graph). The reason is that the attacker attacks with the optimal found attack in round 200, which is probably a very good attack. The attack in this round is the basis of the defender's defense decision. There is only a limited space of very good attacks and those attacks trigger only a limited amount of VM movements. If the attacker attacks with a random strategy in round 200, it tricks the defender into defending against this strategy which is probably not a good decision.

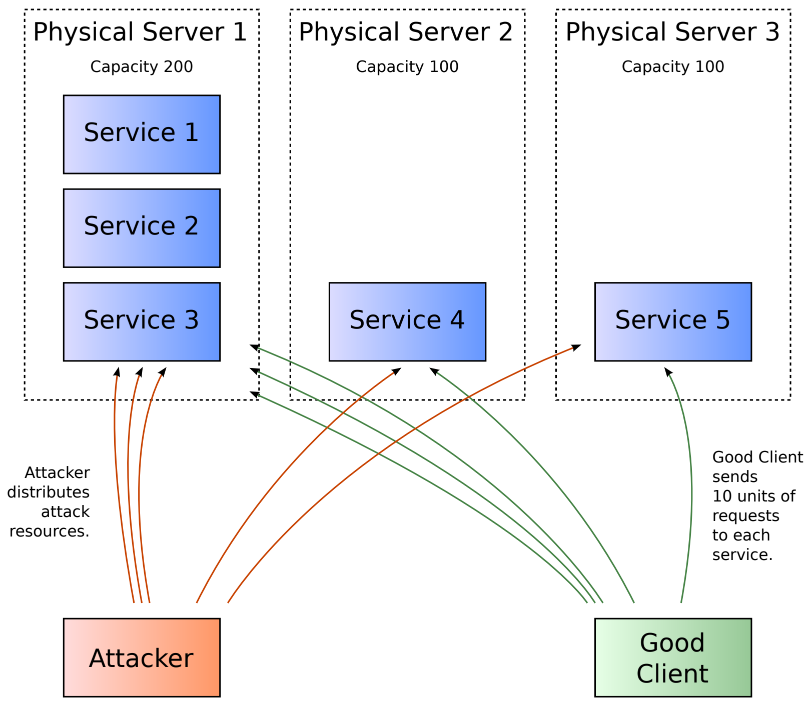


Figure 8.5: Simulation scenario.

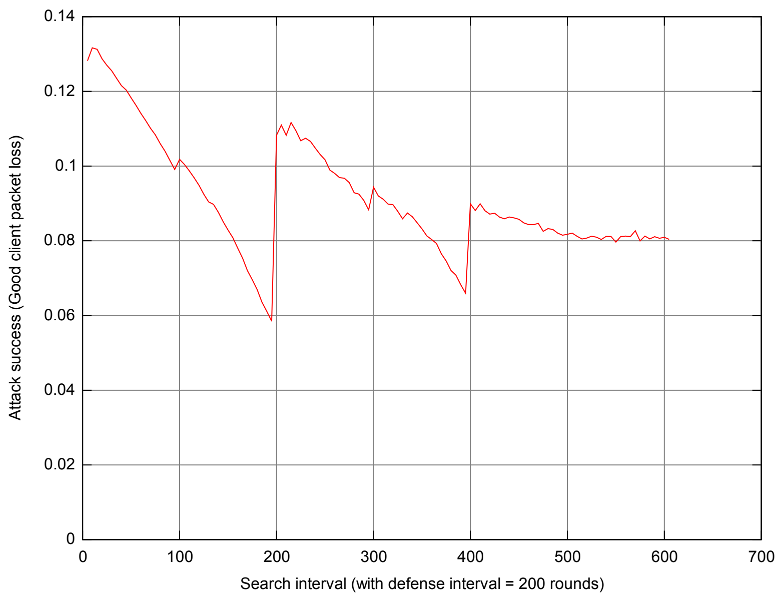


Figure 8.6: *Heuristic attacker vs. bin-packing defender:* Simulation of different attacker search intervals.

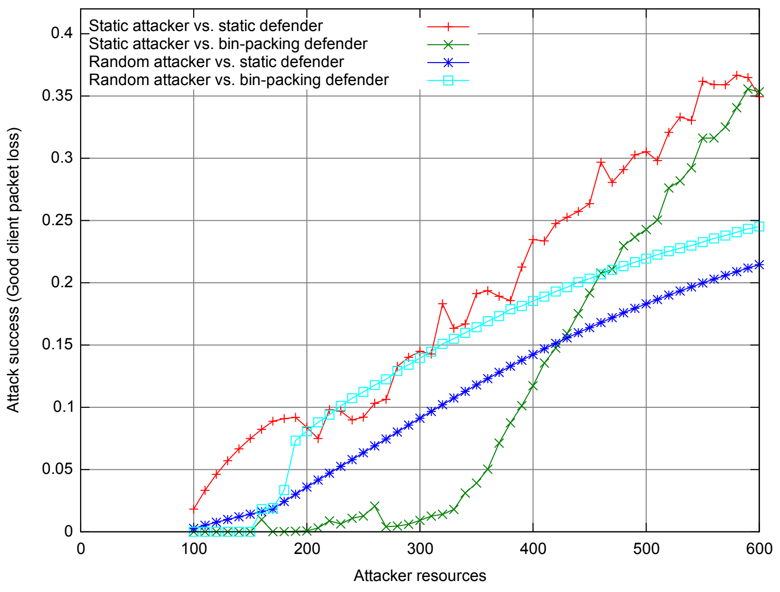


Figure 8.7: Simulation of simple attacker strategies with different attacker strengths.

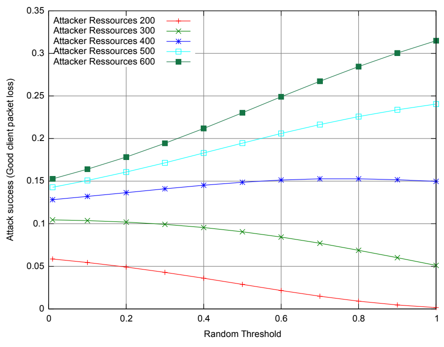


Figure 8.8: Influence of the *random attacker*'s threshold versus a *static defender*.

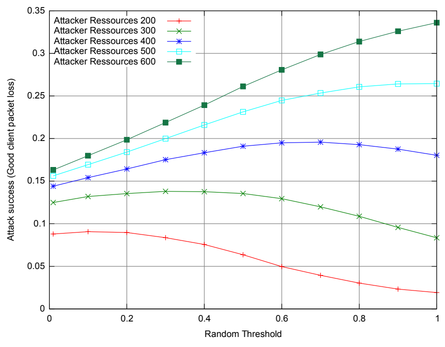


Figure 8.9: Influence of the *random attacker*'s threshold versus a *bin-packing defender*.

Here we see that it is a mistake on the defender's side to base decisions on events that happened in the past if the attacker could observe — or even worse influence — them.

Figure 8.7 shows how the simple attack strategies perform against a *static defender* and a *bin-packing defender*.

The curves of the *static attacker* show random fluctuations. This is because the static attacker only once performs a random search (in the very beginning) and sticks with this decision for a very long time. Even though the curves show averages of 25 simulation runs, these random factors would not converge. All other attackers also make random decisions, but as they do this regularly, the random search happens in the order of 500 times per single simulation run. Therefore all other values show much better convergence.

The static attacker is quite strong when playing against a static defender. We see that a bin-packing defender is much stronger than a static defense. In the end, with

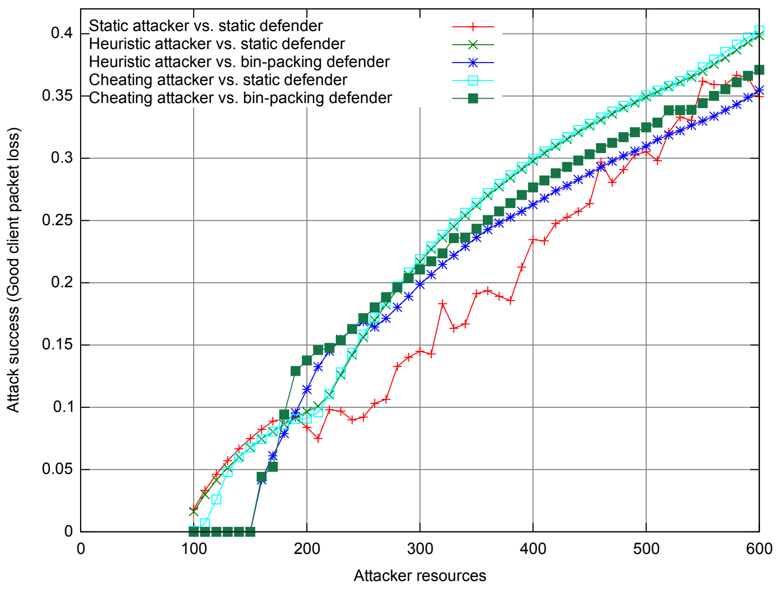


Figure 8.10: Simulation of improved attacker strategies with different attacker strengths.

an attacker strength of 600 the attacker can attack almost anything, therefore a good strategy only brings limited benefits.

The *random attacker* shows a bad performance against the *static defender*. However against the *bin-packing defender* it scores much better. This is because the bin-packing defender will costly move services each 200 rounds, but he will not have any benefit from this as the following attacks will be random again.

In Figures 8.8 and 8.9 we investigate the choice of the random attacker's threshold as discussed in Section 8.4.1 — the dice roll that determines if the random attacker will attack a service or not. Like in Chapter 7 we see that a weak attacker wants to concentrate his attack on few targets while a strong attacker rather wants to spread his attack. Therefore the optimum value shifts towards the right with increasing resources.

As noted in Section 7.3 an attacker wants to choose which servers to attack depending on his resources and then hit all services on the selected servers. However, in our scenario the attacker does not know the assignment of services to servers.

In the simulations we have 3 servers. It will be optimal for a weak attacker to attack a single server. Therefore the limitation to attacking 30% of the services increases the chance that the attacker will select the right services. Therefore the chosen threshold of 30% for all other simulations is beneficial for a weak attacker, i.e. one that has less than 400 resource units available. A strong attacker would rather select a higher value.

Figure 8.10 shows the performance of the improved attack strategies. The *static attacker* versus *static defender* curve is provided again as a reference. We see that *heuristic attacker* and *cheating attacker* both perform well, while the former one is still a bit weaker than the latter one. Versus the *static defender*, both attackers' curves show a dent at 210 attacker resource units. This is the time when the attacker decides to switch from attacking a single target (in this case a physical server) to attacking two. This phenomenon was discussed in Section 7.2, Figure 7.9 shows it as well.

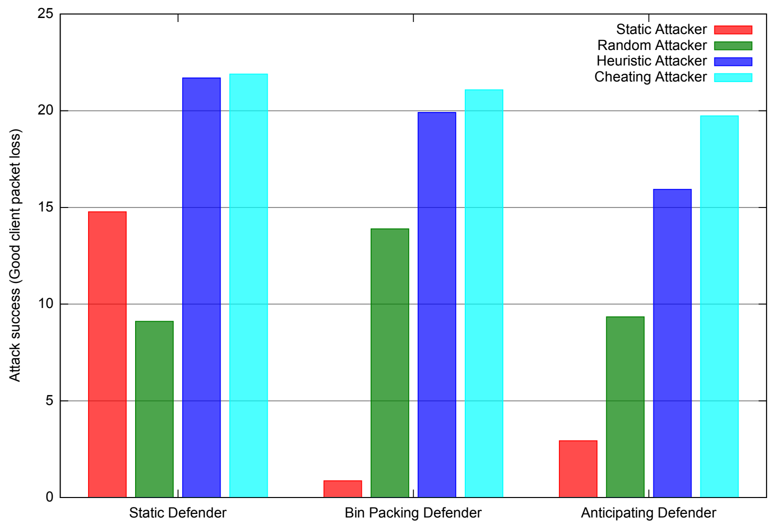


Figure 8.11: Final evaluation including the *anticipating defender*.

Between 180 and 280 units of attacker resources, the *bin packing defender* performs worse than the static defender. This is caused by the overhead of moving virtual machines, possibly this effect could be improved by increasing the VM movement penalty in the defense strategy calculations. Overall we see that the *bin packing defender* reduces the damage caused by the attacks, however against a strong attacker the benefit is not very big.

Until now, the *anticipating defender* was not considered. The fact that an inner and an outer optimization have to be calculated made it impractical to calculate this strategy for the above graphs. Figure 8.11 shows simulation results with attacker resources being set to 300, which includes all attack and defense strategies.

Again we see that the *static defender* is the worst defense strategy against most attacks. It only performs well against the *random attacker*. The *bin packing defender* is very good against static attacks, but there is little gain against optimized attack strategies. It also performs very bad against the random attacker because of unnecessary service movements.

The *anticipating defender* is better than the *bin packing defender* when countering advanced attacks. It also performs about as well as the *static defender* when being attacked by the *random attacker*. Interestingly the *bin packing defender* outperforms the *anticipating defender* when it comes to countering static attacks. This is because the *anticipating defender* sacrifices defense efficiency against the current attack for being less vulnerable in the future. So the *bin packing defender* is more vulnerable, but the *static attacker* is not smart enough to exploit the vulnerabilities.

We can see that the *anticipating defender* performs a bit better than the *bin packing defender*. However, the difference is not large.

8.6 Conclusions

In this chapter we have shown that moving virtual machines can improve the defender's situation when facing a DoS attack. The attacker is free to change his strategy at any moment, but he has no information about the new state of the network and therefore wastes time probing for a new effective DoS strategy. A requirement for an improvement on the defender's side is that the cost of moving the virtual machines is lower than than the loss which the defender suffers while searching for his new attack strategy.

We limit the defender to making choices in fixed intervals. This is an artificial limitation which is meant to reduce the strategy space. Another limitation is the fact that the defender only uses the load distribution in one single round as the basis of his decisions. Better strategies may exist for the attacker as well as the defender. Proving that a certain strategy is optimal for this scenario is probably very hard.

For example, a hypothetical better attacker would not just make a decision which is optimal for the moment, but plan in advance and try to influence the defender's next resource allocation. This would not only require the attacker to have detailed information about the defender's current state, but also knowledge about his decision making. This even better attacker would be strong against a *bin packing defender*, but probably its gain against the *anticipating defender* would be very limited.

It is generally dangerous for the defender in denial-of-service games, to make decisions based on observations from the past in the presence of a smart attacker. The attacker will also observe the basis of the decisions (or even be able to modify it) and therefore predict or influence the decision making. Further examples will be shown in the following Chapter 9.

The methods developed in this chapter are suitable for countering *Flash Crowds* of legitimate requests which overload services. We assume that a *bin packing defender* would perform better in a flash crowd environment than against a DoS attacker, as the flash crowd — despite of strong load fluctuations — does not willingly try to trick the defender algorithm into sub-optimal decisions.

Key findings in Chapter 8:

- **Moving services at runtime brings a great benefit against an attacker who is not aware of this and will therefore not react effectively.**
- **A smart attacker can greatly reduce the benefit of this method.**
- **As a defender, it is dangerous to base decisions on the attacker's past strategy. The performance of the new strategy against its own best response should be the main criterion.**
- **Testbed experiments have shown that VM migration during a DoS attack is practically feasible and can be expected to bring a benefit as long as attackers do not adapt.**

8.7 Acknowledgements

The author would like to thank Young Chul Jung who developed the initial strategies for attacker and defender and Andreas Korsten for the testbed results in Section 8.2.

9. Protection of Webservers using Proxies

People tend not to think about security until it is too late. Adding security to a product or web site takes time and creates additional costs. As long as no attack happens, there is no benefit from having security features. Such views seem short-sighted at first, but depending on the situation it may be rational to ignore security advice, if the costs outweigh the benefits [50].

In case of denial-of-service attacks, it is very helpful if a service scales well when adding additional hardware. But being able to replicate a service to multiple servers usually requires synchronization among the instances which is hard to achieve. Therefore, most web applications are developed in a way that gives no possibility of easily creating replicas and distributing them to multiple servers.

In most cases it is surely possible to change the web application's architecture to make distribution easier, however, this takes time and when the attack is already ongoing it is too late to make major architectural changes.

In this chapter an experimental DoS defense for web servers is developed that can be applied when the attack has already started and that requires very few changes on the actual server. This system requires no JavaScript code on the client side as many similar approaches do, but purely relies on the HTTP standard [51].

After related work is discussed in Section 9.1, the scope of the defense system is defined in Section 9.2, while in Section 9.3 the system itself is described. In the following sections an evaluation specific aspects of the system is given, first theoretically (Section 9.4), then via simulation (Section 9.5) and finally using experiments on a testbed (Section 9.6). The chapter is concluded in Section 9.7.

9.1 Related Work

A number of other publications use proxies between clients and server for countering denial-of-service attacks. This related work has already been discussed in Section 2.3.5.

The novelty in this thesis is the proof-of-work, which is done by redirecting the client various times between the proxies. Therefore our test can be completed by any HTTP-compliant client, there is no need for additional client-side logic.

Different proof-of-work techniques are discussed in Section 2.3.6. The closest competitor to our method are client-side computational puzzles based on JavaScript, e.g. provided by the Apache module *mod_kaPoW* ([19]).

9.2 Scope

As discussed in Section 2.2.2 there are different types of DoS attacks. The most simple ones try to exhaust the victim's bandwidth by sending large amounts of meaningless UDP or ICMP packets. More specific are attacks that try to interact with the victim and lure it into reserving resources.

In recent years, one new type of attack that has been observed made a complete TCP handshake and then sent partial HTTP requests. Optimized web servers that try to process requests fast and therefore reserve resources as early as possible, e.g. the Apache web server, were especially vulnerable against these attacks.

Generally the type of attack which is the hardest to detect is the complete application layer attack. The attacker simply has more resources available than the server and sends complete Layer 7 (e.g. HTTP) requests. There may be web pages on the server that are especially hard to generate (i.e. search results that require complex database queries). In this case the attacker can try to concentrate on such pages to maximize the effect of his attack.

The defense system presented in this section is based on proxies. Therefore all attacks that do not send complete Layer 7 requests will not affect the actual web server but end at the proxy layer. The server is therefore protected, however it might be possible to overload the proxies this way.

In the following we will distinguish two categories of attacks:

- attacks with complete requests that can pass the proxy defense layer and reach the actual web server, and
- attacks that specifically try to overload the proxies. This is possible with flooding attacks if the attacker has enough resources or with partial HTTP requests.

9.3 Approach

A DoS attacker has limited resources at his disposal, however if the attack is successful the attacker's resources are sufficient to make the defender spend all of *his* resources for processing the attacker's requests. Like in some of the related work, our basic ideas are:

- Introduction of an intermediate layer between the clients and the web server, which has more resources and can therefore filter incoming requests. This defense layer is based on proxies that can be set up cheaply and easily at commercial web hosting providers.
- Making the client spend / waste resources before being able to send an actual HTTP request to the server.

The novelty of our system is that the system is completely based on HTTP redirects. Each client has to "prove" that it spent resources by sending its request multiple times to the proxies before it is processed by the actual web server. This idea is based on the assumption that resources on the web server are scarce, but that it is possible to set up a strong proxy defense that can match the attacker's resources.

9.3.1 Using the System to defend against a DoS Attack

The following steps are necessary during a DoS attack to set up the proposed system:

- Set up multiple proxy servers at some web hosting provider. Currently the proxies are written as Java Applications, but this is not a hard requirement, they can easily be ported to other server technologies.
- Inform each proxy about
  - the addresses of all other proxies,
  - the address of the web server to protect and
  - a shared secret — a common symmetric encryption key K_{secret}.
- Change the web server's IP address. The whole proxy layer would be useless if the attacker still knew the real server's IP address allowing him to bypass the proxy layer, therefore the server's address must be changed. We assume that this is possible without too much hassle.

  Ideally the defender's ISP should drop traffic to the old IP address as early as possible in his network. A sophisticated architecture for hiding the server's real location and filtering all traffic to the server that does not originate from a proxy layer similar to ours has been described in [15].
- The target domain's DNS server has to be configured to return proxy IP addresses in a round-robin fashion when being asked about the web server's IP address.

After these steps the system is operational and can process request from clients as described in the following Section 9.3.2.

9.3.2 Processing a Client's Request

Figure 9.1 shows the message flow of a client requesting a web site while the protection is active.

1. The client queries DNS for the web server's IP address.
2. DNS returns the IP address of one of the proxies.
3. The client sends its HTTP-request to the given proxy.
4. The proxy now generates an HTTP redirect message with status code `302 Found`. This message redirects the client to the same URL at another randomly selected proxy. The redirected URL also contains an additional HTTP GET parameter `pcode` with encrypted information, the main part of which is the number of already performed redirects (in this case 1) and a timestamp. The complete content of this parameter will be discussed in Section 9.4.4.

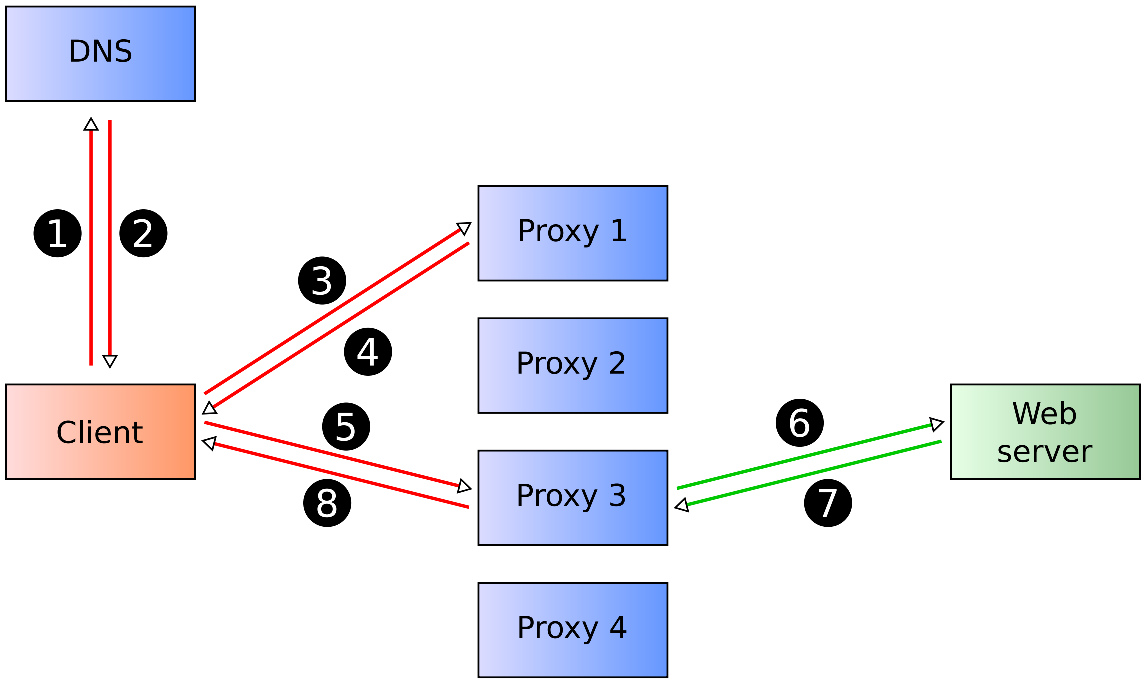


Figure 9.1: Message Flow with DoS defense.

5. The client receives the redirect and sends its request again to the given proxy. The new request includes the added parameter which the client could not read or modify because of the encryption.

6. The new proxy receives the request and evaluates the `pcode` parameter. The number of already performed redirects is compared to a target number (one in the example).

7. If the request has been redirected often enough, it is passed to the web server for processing. The `pcode` parameter is removed from the URL, so the web server sees the original request from the client.

8. The web server processes the request as it normally would and sends the reply back to the proxy.

9. The proxy passes the reply to the client.

A message sequence chart of the communication is shown in Figure 9.2. With n being the number of redirects that the client has to go through, one can see that a potential attacker has to send $n+1$ requests for getting one request to the web server if there is no way to bypass the system. Therefore the attack would be attenuated by a factor of $\frac{1}{n+1}$.

It is clear that the proxies need enough resources to cope with the full attack capacity of the attacker. The assumption here is that proxies are relatively cheap and that new ones can be created easily. The only purpose of the proposed system is a reduction in the number of attack requests to the actual server. The following sections show how effectively this goal can be achieved.

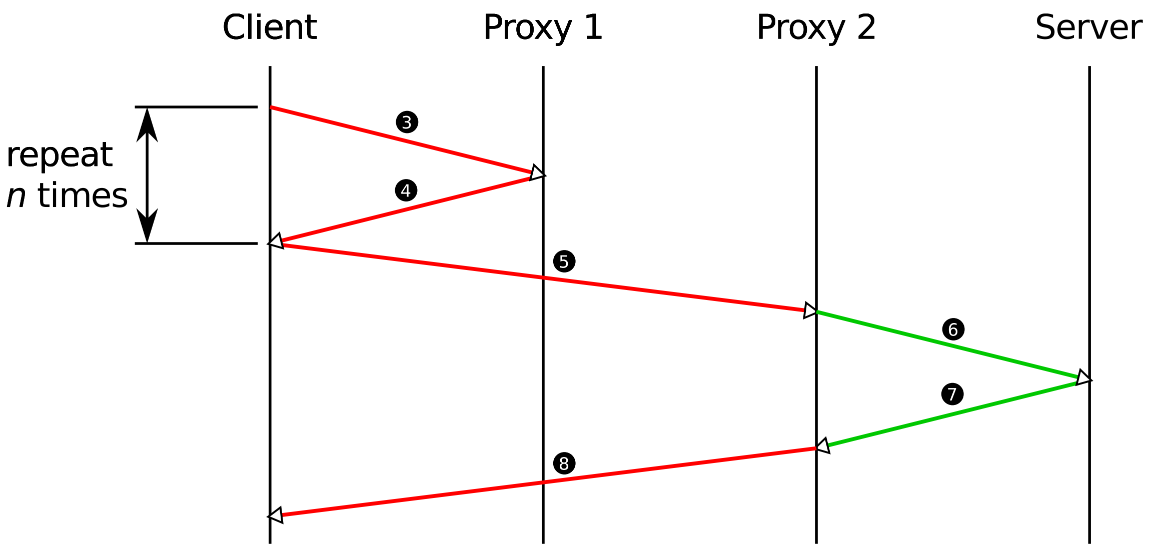


Figure 9.2: Message sequence chart.

The proposed system is only a good choice for dynamic content that is generated specifically for the client. Purely static web sites, images and other documents can be cached by the proxies. As caching proxies are a technology that has been in use for a long time, this case is not explicitly investigated in the following. We assume that the proxies are able to distinguish between static and dynamic content and that each proxy fetches static content only once and responds to later requests by serving a cached copy. Therefore we only have to consider dynamic content in the following.

9.4 Analysis of the System

In this section we analyze potential weaknesses of the proposed system. This analysis will be extended by simulations in the following Section 9.5.

9.4.1 Replay Attacks

A smart attacker would try to bypass the system by replaying message (5) in Figure 9.1. This message will be called *Ticket Message* in the following text, as it ultimately permits access to the service.

Some basic replay protection can be provided by adding a timestamp to the `pcode` parameter. Assuming synchronous clocks on all proxies, it is possible to define a lifetime for the ticket. Replaying the ticket message is only possible during that lifetime, outdated ticket messages are dropped by the proxy. This section discusses the degree of protection that this anti-replay mechanism provides.

From the attacker's perspective, the situation looks as follows: First he needs to spend a certain amount of effort on obtaining a ticket message. Then this ticket can be used for a limited time to send requests directly to the web server. During this time the attacker is only limited by its own capacity and the capacity of the relaying proxy. Figure 9.3 shows a time-sequence chart of such an attack.

For the theoretical investigation of this attack's consequences, we make the following assumptions:

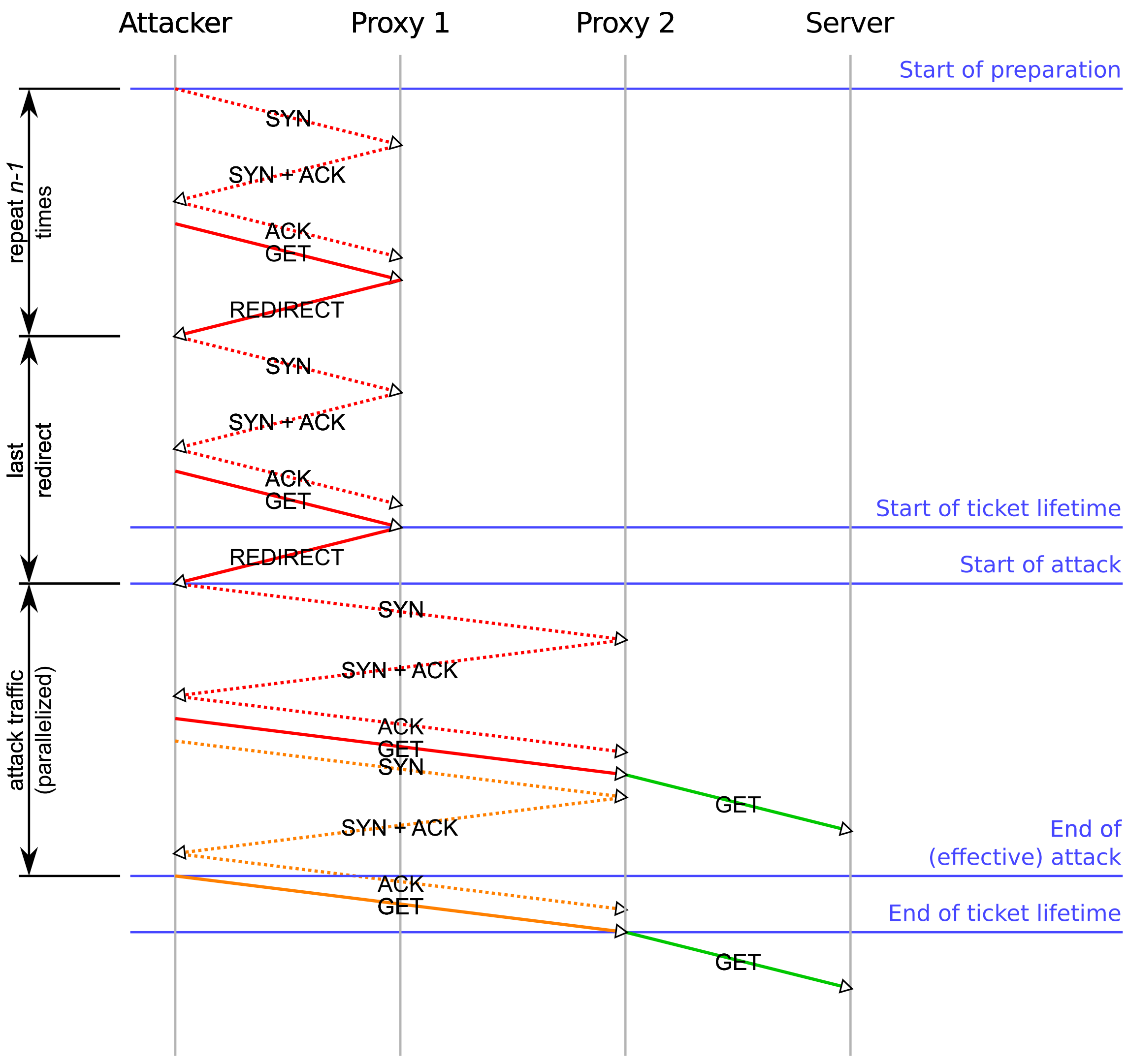


Figure 9.3: Message sequence chart of a replay attack.

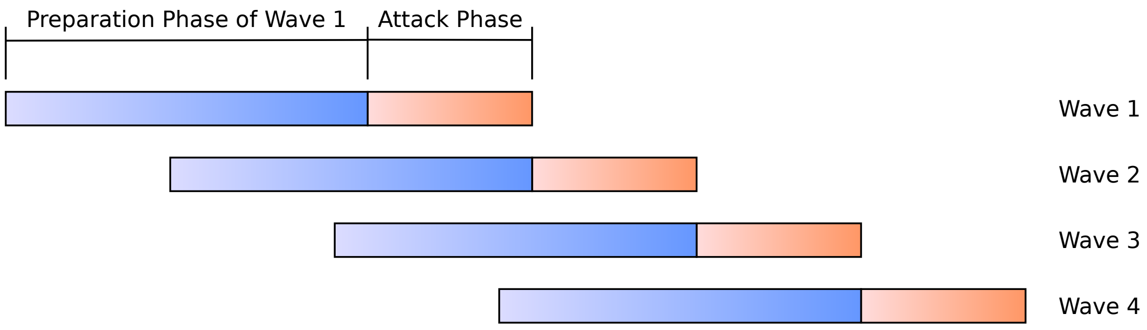


Figure 9.4: Pipelining the attack by obtaining ticket messages in the background.

- As we want to investigate the worst-case traffic that the attacker can send, we do not consider any resource constraints on the webserver or the proxies.
- We assume an HTTP 1.0-style behavior on the proxies. This means that the HTTP connection to a client is closed as soon as a single request has been answered. This introduces additional overhead compared to modern HTTP 1.1 "keep-alive" connections. We make this choice as this extra overhead affects the client and the proxies and not the web server we are trying to protect, therefore it works in our favor.
- We assume that the attacker wants to run a continuous attack for a long time, so he has to obtain new ticket messages in the background while sending attack traffic. Therefore he has to pipeline his attack in waves as shown in Figure 9.4. Ticket messages are obtained during a "preparation phase" and used during the actual "attack phase". The number of parallel waves is chosen to make sure that at every time, one wave is in its attack phase.

For the investigation, we need the following parameters:

- b_o outgoing link capacity of attacker in bits per second
- b_i incoming link capacity of attacker in bits per second
- $S_o = S_{SYN} + S_{ACK} + S_{GET}$, size of the attacker's outgoing messages for one request in bits
- $S_{i,p} = S_{SYNACK} + S_{REDIRECT}$, size of the reply-messages from the proxy to the attacker during the preparation phase in bits
- $S_{i,a} = S_{SYNACK} + S_{REPLY}$, size of the reply-messages from the proxy to the attacker during the attack phase in bits
- l ticket lifetime in seconds.
- t attack phase duration, $t = l - RTT$ in seconds
- n Number of redirects

From these values we want to calculate the *request rate* r, the number of requests per second that the attacker can send to the actual server.

Without any defense, the attacker can send

$$r_{unconstrained,out} = \frac{b_o}{S_o} \left[\frac{requests}{s}\right].$$

With the defense mechanism as described above, the attacker has to go through a preparation phase before being able to send requests to the server, like shown in Figure 9.3. We assume that the attacker prepares the next attack phase during the current attack phase to make sure that the victim is permanently unreachable.

In this case we can give a first approximation for r as

$$r_{defense,out} = \frac{\frac{t \cdot b_o - n \cdot S_o}{S_o}}{t} \left[\frac{requests}{s}\right].$$

The sum of the available link capacity during one attack phase duration is $t \cdot b_o$. Prior to the current attack phase, the attacker had to spend $n \cdot S_o$ units of bandwidth on the preparation. As attacks are pipelined and new preparation phases happen in parallel to the current attack, we can assume that this is the amount of bandwidth that has to be spent for future attacks during the current attack phase. So the available bandwidth for attacking is only $t \cdot b_o - n \cdot S_o$ which reduces the number of requests that are sent during the attack phase to $\frac{t \cdot b_o - n \cdot S_o}{S_o}$.

In the above formula, if n becomes sufficiently big, the attacker wastes all of his bandwidth in preparing and is unable to send all attack packets that he got tickets for. In this case the attacker will have to reduce the attack traffic as well as the preparation traffic to a certain fraction, so his problem is to optimize

$$r_{defense,out,lower} = \max_f \frac{f \cdot \frac{t \cdot b_o - f \cdot n \cdot S_o}{S_o}}{t} \left[\frac{requests}{s}\right].$$

This formula is a lower bound. It can be justified by separating the attack phases into busy and idle attack phases. The later ones are caused by the fact that the attacker sends less preparation traffic and therefore does not have sufficient tickets for a continuous attack.

The assumption that makes this formula a lower bound is that during these idle attack phases, there is no attack traffic (since the attacker does not have a ticket) while the preparation traffic is a long-term action which has to be the same during busy and idle attack phases. The remaining bandwidth during idle attack phases is simply not used by the attacker.

To calculate an upper bound, we look at the attacker's constraints. Again the attacker chooses a parameter f which defines the ratio of attack phases where attack traffic is sent and idle attack phases where no ticket is available to send traffic. Averaging over a longer period of time, we can formulate two constraints:

- A Bandwidth constraint, only the non-preparation bandwidth is available to send attack traffic

$$request\_rate_{defense,out,upper,bw} = \frac{\frac{t \cdot b_o - f \cdot n \cdot S_o}{S_o}}{t}$$

- and a ticket constraint, as the attacker can only send attack traffic if he has a ticket for the round

$$request\_rate_{defense,out,upper,tickets} = \frac{\frac{t \cdot b_o \cdot f}{S_o}}{t}.$$

With larger f, $request\_rate_{defense,out,upper,bw}$ will decrease. At the same time, $request\_rate_{defense,out,up}$ will increase. One has to find an optimal f that maximizes the minimum of both formulas. Setting

$$request\_rate_{defense,out,upper,bw} = request\_rate_{defense,out,upper,tickets}$$

gives

$$f = \frac{b_o \cdot t}{b_o \cdot t + n \cdot S_o}.$$

Solving for the request rate, it follows

$$\begin{aligned} request\_rate_{defense,out,upper} &= \frac{\frac{t \cdot b_o - \frac{b_o \cdot t}{b_o \cdot t + n \cdot S_o \cdot n \cdot S_o} S_o}{}}{t} \\ &= \frac{b_o^2 \cdot t}{b_o \cdot S_o \cdot t + n \cdot {S_o}^2} \end{aligned}$$

which is our upper bound for r.

Without any replay attacks, the attacker has to go through a full preparation phase for each single request that he sends. Therefore the rate of outgoing traffic that actually reaches the server is reduced to

$$request\_rate_{perfectdefense,out} = \frac{b_o}{(n+1) \cdot S_o} \left[\frac{requests}{s}\right].$$

With this attack, the attacker has to be able to react to incoming replies, otherwise he will not be able to obtain new tickets. Therefore he must be careful not to DoS his own incoming bandwidth. This might be the harder constraint, as HTTP responses that actually include content are usually much larger than the corresponding HTTP requests. On the other hand, many Internet access technologies are asymmetric (e.g. ADSL). Therefore, if the attack-traffic is sent from a botnet, we would commonly have $b_i > b_o$.

Analogous to the above considerations it is possible to derive similar formulas for incoming traffic. This is omitted here as they are not needed for the following argumentation.

9.4.2 Evaluation of lifetime-based Replay-Protection

Figure 9.5 shows the reduction of traffic to the server when applying the discussed defense mechanism. The following parameters were chosen:

Parameter	Value		Parameter	Value	
b_o	1.0	MBit/s	S_{SYN}	40	Bytes
RTT	0.1	s	S_{ACK}	40	Bytes
			S_{GET}	400	Bytes

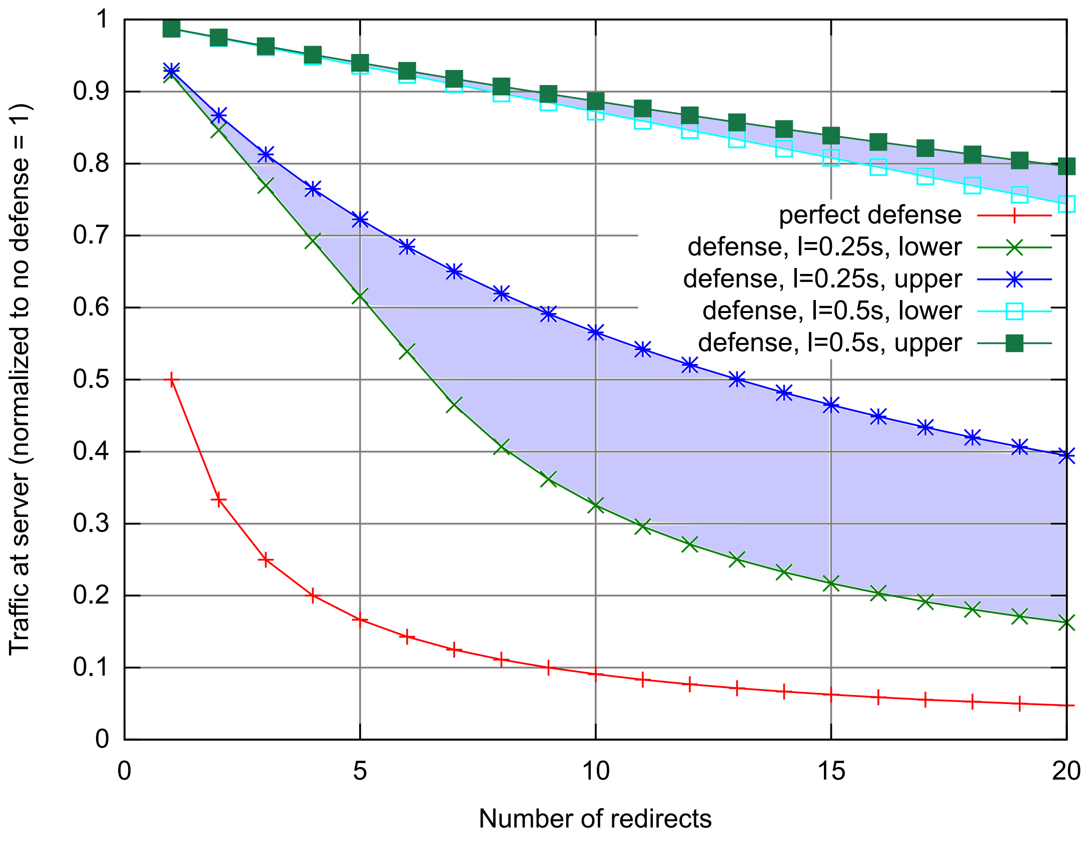


Figure 9.5: Effectiveness of the defense.

As seen in Figure 9.3, the lifetime of the actual ticket has to be at least two RTTs long. Internet services should generally be accessible from anywhere on the globe. Customers who are located close to the service generally experience short RTTs (e.g. 20 ms from Munich to servers in Germany). RTTs get longer the further customer and service are apart (e.g. 300 ms from Munich to South Korea). The type of Internet connection also plays a role, a modem connection also increases the delay for each packet.

Considering this, 0.5 seconds would be a low value for the lifetime, even without the presence of a DoS attack that slows down packet processing in routers and on the proxies.

Practically, up to 20 redirects are possible. With a higher number, the web browser will no longer redirect but display an error message.

From Figure 9.5 we can see, that under these circumstances, we can not hope to reduce the DoS traffic on the server by more than 20%. Therefore the replay attack makes the purely lifetime-based tickets unusable.

9.4.3 Protection from Replay Attacks

It is possible to achieve perfect replay protection when storing some data per connection attempt. For this two conditions must be fulfilled:

- *Make sure that the same proxy does not forward the same request twice to the server.* This can be done by incorporating some unique ID into the requests and making each proxy store a list of IDs that were recently forwarded to the server.
- *Make sure that the same request is not forwarded to the server by two different proxies.* One method of doing this is to let the first proxy that receives a new request decide which proxy will be the last one in the chain. This information also has to be stored in the `pcode` encrypted HTTP parameter. If a proxy receives a request with enough redirects but if he is not the one who is allowed to forward this request to the server, then this request will be dropped.

The request ID can be of the form "<ID-Assigning-Proxy>-<Counter>" to be unique in the system (or at least unique for the duration of one connection request).

This scheme requires each proxy to keep some state (a list of recently forwarded IDs), which we initially wanted to avoid. However the state is only in the order of a few bytes per request and can be expired based on timestamps regularly. As the purely lifetime-based replay protection has been shown to be ineffective in Section 9.4.1, ID-based replay protection was used in the actual implementation.

9.4.4 Content of the encrypted Parameter

Functionally, the `pcode` parameter has to include the following information:

- The number of remaining redirects.
- A request ID assigned by the first proxy in the chain.
- The ID of the final proxy that is allowed to send the request to the server.

Algorithm 2: Proxy forwarding logic

```
input: $R$: incoming HTTP request, $n$: target number of redirects,
       $k$: symmetric_encryption_key, $counter$: ID-space for messages,
       $recently\_forwarded\_tickets$: message IDs that have passed through
       this proxy lately
if not $R.pcode$ exists then
    $exit\_proxy = choose\_random\_proxy()$
    $next\_proxy = choose\_random\_proxy()$
    $num\_redirects = 1$
    $request\_ID = self.proxy\_ID|``-'' |counter$
    $counter = counter + 1$
    $R.pcode =$
      $Enc_k(IV|request\_ID|num\_redirects|exit\_proxy|integrity\_info)$
    $redirect(R, next\_proxy)$
else //pcode existed already
    if not $check\_integrity(R.pcode)$ then
        $drop(R)$
    $request\_ID, num\_redirects, exit\_proxy = Dec_k(R.pcode)$
    if $num\_redirects < n - 1$ then
        $next\_proxy = choose\_random\_proxy()$
        $num\_redirects = num\_redirects + 1$
        $R.pcode =$
          $Enc_k(IV|request\_ID|num\_redirects|exit\_proxy|integrity\_info)$
        $redirect(R, next\_proxy)$
    else if $num\_redirects = n - 1$ then
        $num\_redirects = num\_redirects + 1$
        $R.pcode =$
          $Enc_k(IV|request\_ID|num\_redirects|exit\_proxy|integrity\_info)$
        $redirect(R, exit\_proxy)$
    else if $num\_redirects = n$ then
        if $exit\_proxy = self.proxy\_ID$ and
        $request\_ID \notin recently\_forwarded\_tickets$ then
            $recently\_forwarded\_tickets \leftarrow request\_ID$
            $remove\_pcode(R)$
            $forward\_to\_server(R)$
        else // wrong exit proxy or duplicate
            $drop(R)$
    else // invalid num_redirects
        $drop(R)$
```

- Optionally a timestamp can be used to detect outdated requests. However this purpose can partially be fulfilled by the message ID.

To avoid manipulation, some standard cryptographic additions are used:

- An initialization vector that prevents identical plaintexts from producing identical ciphertexts. This can be omitted if plaintexts are sufficiently different or a shortened IV can be transmitted (i.e. by using an IV that is shorter than the used encryption algorithm's block size and expanding it via some expansion function).
- Some integrity verification mechanism, i.e. an HMAC value of the encoded message. Otherwise the attacker would be able to blindly change the ciphertext and test what happens.

The whole procedure is summarized in Algorithm 2. The encoded numbers $request\_ID$, $num\_redirects$ and $exit\_proxy$ easily fit into one AES block of 128 Bits. IV and hash are each another 128 Bits long.

Finally, the `pcode` parameter has to be encoded in a printable way to be usable as part of an URL. Using Base64, 384 Bits of raw data translate to 64 characters in the URL (67 characters if the format is &p=<content>).

9.4.5 DoS Weakness of the Scheme

A smart attacker will try to find weaknesses in any defense scheme and unfortunately, this DDoS defense scheme has one major weakness.

An attacker could focus his attack on a single proxy. Because requests have to pass through a chain of proxies and each of these proxies has to be responsive at the time, such a DoS attack will affect a large percentage of the user requests. For example with 20 proxies, and 15 redirects to uniformly chosen proxies, about 56% of all connections would be affected if a single proxy was unavailable.

An attack on one of the proxies is also much easier than an attack on the server, as the attacker does not have to respond to redirects — standard UDP flooding would be sufficient.

Possible solutions for this are:

- A proxy always redirects to himself. This would greatly reduce the fraction of affected connections. When receiving IP addresses of multiple proxies by DNS, the client could also try a different proxy if the first one does not respond.

  The core idea of our DoS defense scheme does not require forwarding of requests from one proxy to another, forwarding to the same proxy would also be sufficient. Forwarding between the proxies was initially only introduced for finer load-balancing, which is a minor advantage.
- Apply a load-balancing scheme among the proxies by always redirecting to proxies that currently have a low load. This requires the exchange of load information among the proxies in the background.

  Using this option, one has to make sure that load balancing does not make the load oscillate (a proxy is flooded with requests after reporting low load) and that the load balancing can not be exploited by the attacker.

Simulating these aspects will be subject of the following section.

9.5 Simulation

Load-balancing issues and possibilities to exploit them were investigated simulatively. The simulation is round-based and contains two clients (good and evil) and a configurable number of proxies. In each round, all clients send requests to the proxies first, then the proxies respond if they are not overloaded. The response may contain a redirect, in this case the client is free to decide whether it will follow it. No replay attacks were possible in the simulation, the assumption is that this is prevented by the scheme described in Section 9.4.3.

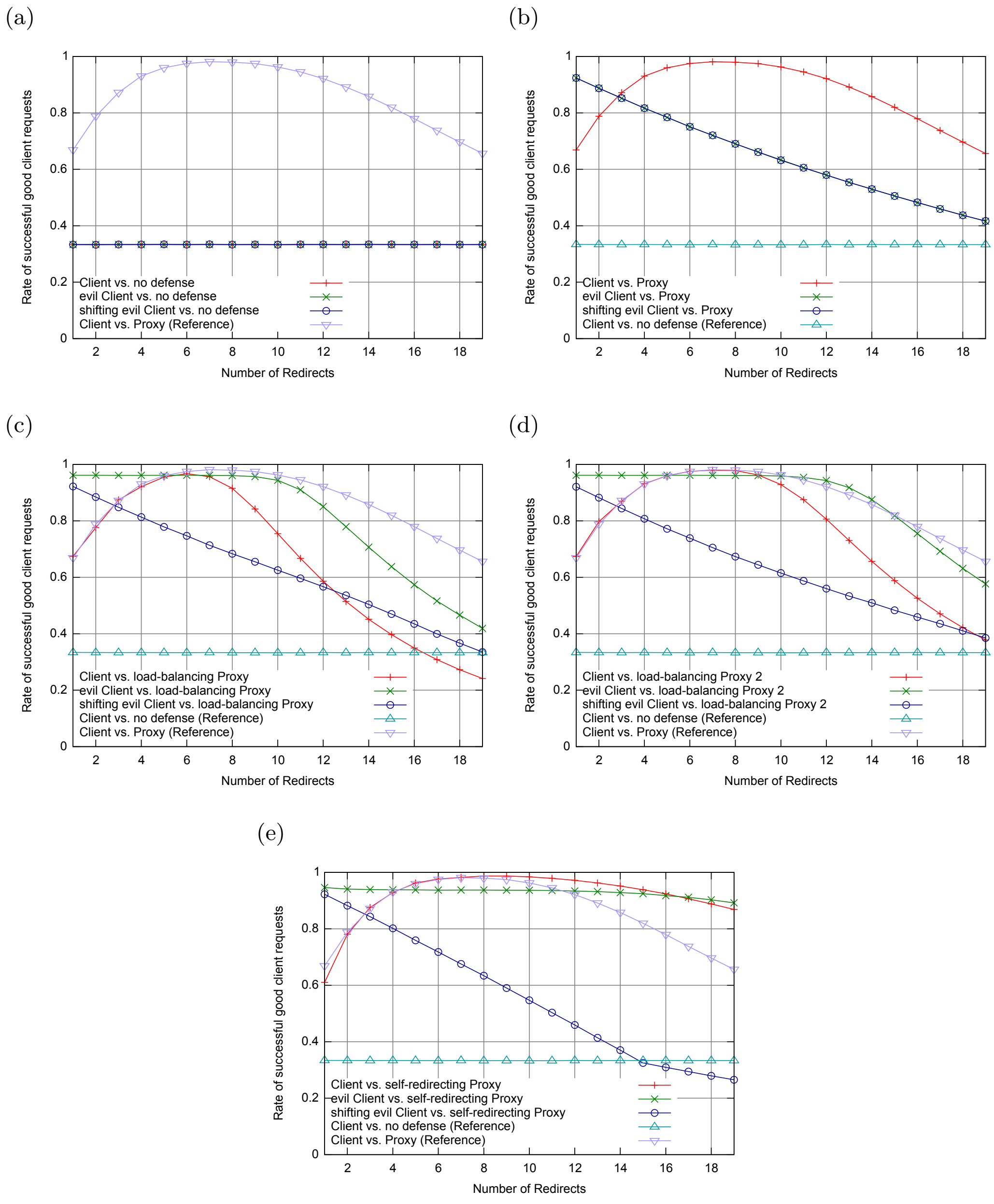


Figure 9.6: Simulation of different attackers and proxies. Simulation parameters: Good Client capacity 5, server capacity 10, single proxy capacity 10, number of proxies 16, attacker capacity 25, 20000 simulation rounds.

As in previous chapters, the rate of successful requests of a good client is used as the primary metric for the attacker's and defender's success. While the good client

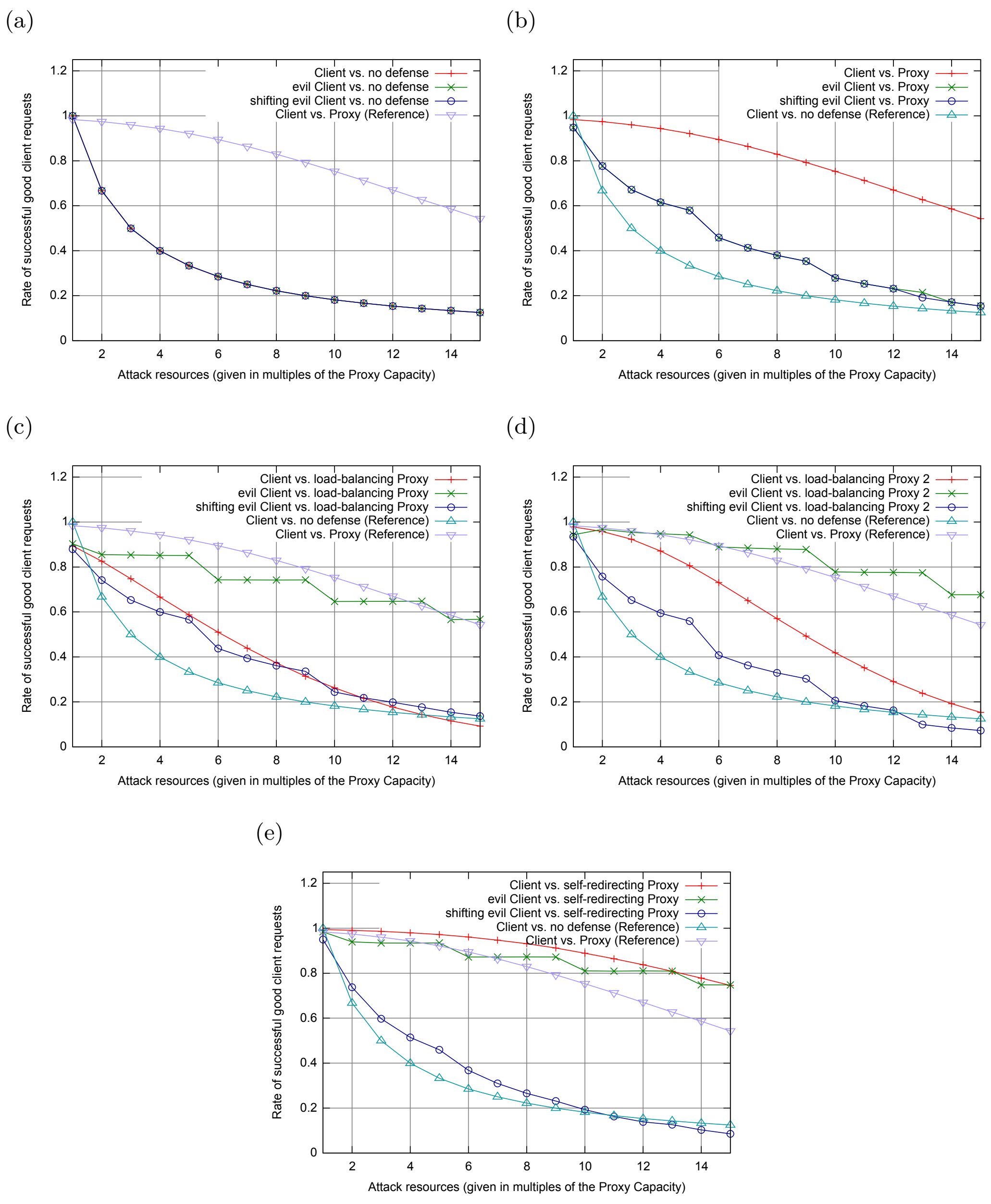
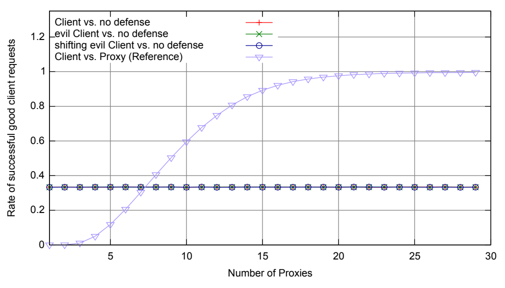


Figure 9.7: Simulation of different attackers and proxies. Simulation parameters: Good Client capacity 5, server capacity 10, single proxy capacity 10, number of proxies 16, number of redirects 12, 20000 simulation rounds.

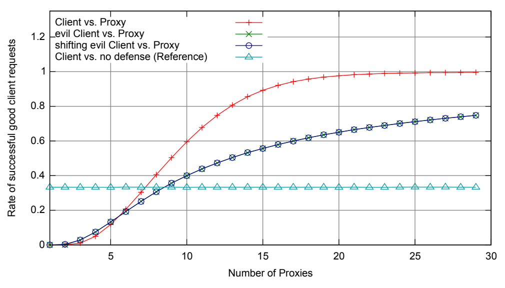
always chooses a random proxy for his first requests and then follows the chain of redirects, different variants of the evil client have been implemented:

- A simple attacker *client* that behaves the same way as the good client, trying to follow redirects to increase the load on the proxies as well as on the server.
- An *evil client* that focuses his resources on attacking the proxies, exploiting the weakness described in section 9.4.5. It first calculates the number of proxies to attack for optimal packet loss based on the ratio of his own resources to the target's available resources. A discussion of optimally selecting the number of targets can be found in Section 7.2.

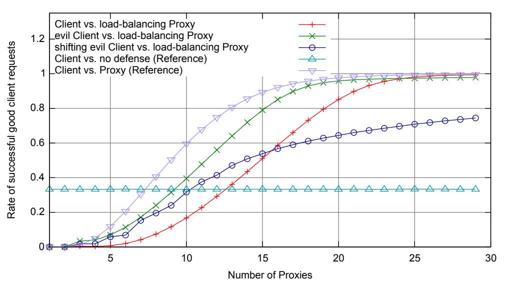
(a)



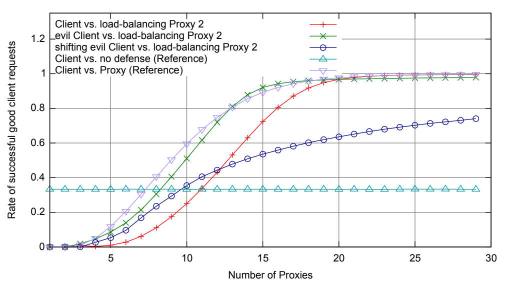
(b)



(c)



(d)



(e)

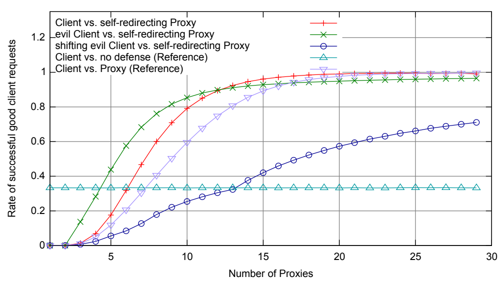


Figure 9.8: Simulation of different attackers and proxies. Simulation parameters: Good Client capacity 5, server capacity 10, single proxy capacity 10, attacker capacity 25, number of redirects 12, 20000 simulation rounds.

This number of proxies is then flooded with requests. The evil client does not follow redirects but sends new requests in each round and it always floods the same proxies.

On the other hand different proxy strategies are compared:

- A *randomly redirecting proxy* which always chooses the target of the next redirect uniformly among all proxies (including itself). This basic strategy was assumed in all previous sections.
- A *load-balancing proxy* which is informed about the load of all proxies. It calculates a probability distribution for selecting the target of a redirect from

this load information, each proxy i has a probability of $\frac{free\_resources(i)}{\sum\limits_{j \in proxies} free\_resources(j)}$ of being selected.

We can already assume that this proxy has a potential problem, which might in reality be exploited by an attacker. The distribution of redirects happens based on load values that were observed in the past and does not necessarily have any informative value for the future.

In the simulation this might lead to oscillations, as a simple calculation shows. Assume that we have 10 proxies p_i, each one with a capacity of 1.0 resource units and that we have to serve a total of 7.9 load units of requests. p_1 to p_9 are each using 0.8 resource units (1.0 = 100% capacity of a single proxy), while p_{10} is only at 0.7 load due to random fluctuations.

When making a single redirect decision according to the formula above, p_1 to p_9 will be selected with a probability of 9.5%, while p_{10} will be selected with a probability of 14.3%.

Therefore in the next round, the load distribution is different. p_1 to p_9 will have to serve 0.75 load units each ($7.9 \cdot 9.5\%$), while p_{10} will be overloaded with 1.11 resource units.

An attacker can exploit this property for amplifying his attacks by constantly switching his targets.

- An alternative load balancing proxy called *load-balancing proxy 2* in the following is also informed about the load of all proxies. However, this information is only used to avoid redirects to already overloaded proxies. All proxies that are below 100% load are selected with the same probability.

- A *self-redirecting proxy*. As with all proxy variants, the decision which proxy is initially contacted is left to the client e.g. by selecting one of multiple proxies returned by the DNS-query. After that the self-redirecting proxy will always redirect clients only to himself.

  Again, This scheme has exploitable properties. The attacker knows that all requests have to be redirected a certain number of times by the same proxy, so it is sufficient for him to attack each proxy in certain time intervals to drop the legitimate client's requests.

These considerations are taken as the motivation for introducing another attacker type:

- A *shifting evil client* that behaves the same way as the evil client, but with one modification. The client does not always attack the same proxies but shifts his attack in each round. For example if the optimal number of proxies to attack is 3, this client will attack proxies 1-3 in round one, proxies 4-6 in round two and so on.

  This strategy can be expected to be good against the load balancing proxy as it uses the lag of load information to make sure that the attacked proxies have a high load of good client requests.

  It can also be expected to be good against the self-redirecting proxy, as all client requests have to pass an attacked proxy one or even multiple times.

Figure 9.6 shows simulation results. In each graph the horizontal axis shows the number of redirects, while the vertical axis shows the good client's success rate. The simulations contain random decisions that affect the results, however the simulated time was chosen so long that random fluctuations are insignificant. There are five graphs in the figure, each of them shows the performance of one defense strategy against all attack strategies. To make the graphs easier to compare, the "client vs. no defense" and "client vs. proxy" curves are present in all graphs.

As seen in graph (a), without defense the good client's success rate with the given parameters is 33%. Introducing proxies (b) can generally enhance this value. The *client* type of attacker causes a trade-off between a high load on the server (when the number of redirects is low) and a high load on the proxies (when the number of redirects is high). The more sophisticated attackers only target the proxies, so this trade-off is not present.

Figure 9.6 (c) shows that *load-balancing proxy* performs well against the (static) *evil client*, but very badly against the *client* and the *shifting evil client*. While the latter confirms our theory about exploitable weaknesses of the *load-balancing proxy*, the former shows that fluctuations occur even with a high non-attack load and therefore load-balancing in this form is unusable. The alternative *load-balancing proxy 2* performs better than the previous load-balancing, but principally shows the same weaknesses.

The *self-redirecting proxy* is good against *client* and static *evil client*, but even more susceptible against the *shifting evil client* than the other variants.

As expected the *shifting evil client* is the strongest attacker. The standard *proxy* without any additional defense mechanisms is the most robust against this best-possible attack.

In Figure 9.7, each graph shows the strength of the attacker on the horizontal axis, while the vertical axis is the ratio of successful good client requests. The five graphs are equivalent to their counterparts in Figure 9.6. The steps in the *evil client* and *shifting evil client* curves are caused by the attackers choice of targets: When more resources are available, more targets will be attacked.

Again we can see that the *shifting evil client* is the strongest attacker and that the *proxy* without special countermeasures performs best against it.

Especially interesting is Figure 9.8 which shows that the proxy-based defense mechanism can mitigate attacks very well if the defender has enough resources available. With 29 proxies (which is about 6 times the attacker's capacity), the defender is able to serve 75% of the requests even against the *shifting evil client*.

Concluding we see from the simulation that the weakness of redirect-based DoS protection is the fact that the proxies are chained, all redirects have to work for a request to succeed. This reduces success rates significantly even for moderate error-rates. However, if the defender has enough resources at his disposal, decent success rates are possible even against an attacker that tries to exploit this.

Load-balancing schemes do not work here as they always have to rely on information from the past. A smart attacker will always be able to guess, what the system has observed and how it will react. He will therefore be able to attack where it hurts the most. This is why load-balancing makes things worse instead of better in the presence of a smart attacker.

In a real-life system, the existence of queues might make load-balancing more useful than it was in this simplistic model. The *load-balancing proxy 2* could be expected to exclude temporarily overloaded proxies from the chains until they have served the requests in their queues — an effect that was not considered here. This is why the unbalanced *proxy* and the *load-balancing proxy 2* will be tested practically in the following section.

9.6 Testbed Experiments

Until now the investigation of proxy strategies was relatively simplistic. The used simulation tool did not capture timing effects or TCP handshakes.

For evaluation purposes, the standard *proxy* and the *load-balancing proxy 2* were implemented in Java and tested using real systems. A testbed consisting of 12 PCs with Intel Atom CPUs that were running Ubuntu Linux was available for the experiments. As attacker, the *Slowloris* tool was used[1], a perl-script which sends partial HTTP requests. *Slowloris* has shown to be effective against the web server (an Apache 2.2 in default configuration) as well as the proxies. In both cases, the number of open connections is limited and a connection attempt from *Slowloris* blocks a "slot" until a timeout occurs. With the Apache's default timeout value of 5 minutes, already a relatively mild attack can have devastating effects.

We tried to defend against this attack using the proxy without any load balancing and the *"load balancing proxy 2"*. The latter one monitors the load of all other proxies and only forwards to proxies that are not overloaded. There is no modification of forwarding probabilities based on the load, if k proxies are not overloaded, each one will be selected with a probability of $\frac{1}{k}$.

Figure 9.9 (a) shows a simulation using the framework from the previous Section 9.5, which predicts the result of the experiment. There is no curve for "no defense" as we do not expect the web server to send any replies during the attack.

In the scenarios with defense, there are four proxies that try to protect the web server. One of these proxies is attacked with *Slowloris*. Again we expect *slowloris* to completely DoS the attacked proxy, but as it sends partial HTTP requests and the proxy does not forward a request before it is completed, the actual web server will not be affected.

Under ideal conditions and using the *"load balancing proxy 2"*, whether a good client's request will be dropped or not completely depends on the client's choice of the first proxy. If the attacked proxy is chosen, the request is dropped, otherwise it will be served — so 75% of the requests will be served successfully. As the attacker does not follow redirects, there is no way of improving the results by redirecting multiple times — in fact the results will get worse as soon as the number of redirects increases the load of the not-attacked proxies.

In the experiment, the good client script tries to send 600 requests within 2 minutes and 40 seconds. Without an attack, this amount of requests can easily be handled by the web server. In Figure 9.9 (b), the percentage of successfully answered requests based on the total number of 600 requests is shown. It can be seen that except for measurement errors, the general form of the curves is similar to the expectation. The *proxy without load balancing* has an exponentially decreasing success probability,

[1] `http://ha.ckers.org/slowloris/`

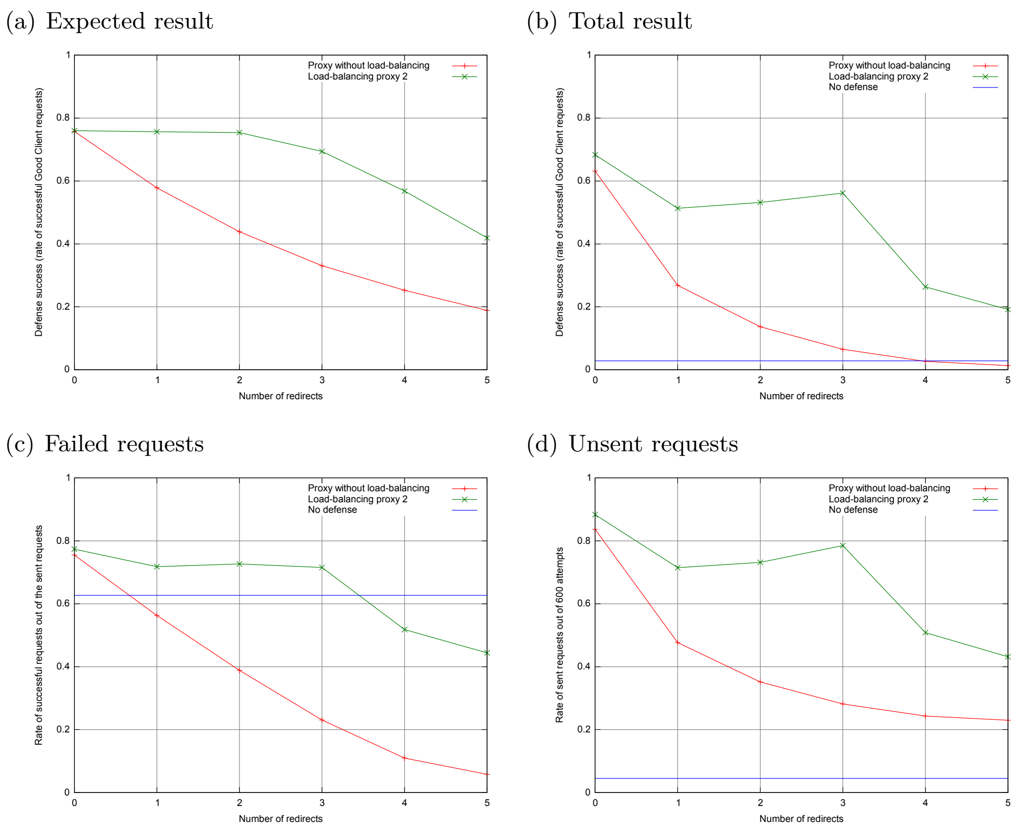


Figure 9.9: Comparison of the proxy system's expected performance with real-world experiments.

with each new redirect there is a probability of $\frac{1}{4}$ that the overloaded proxy is chosen and the request remains unanswered. The *load balancing proxy 2* correctly identifies the situation and therefore performs much better, however, at four and five redirects, the load of the whole proxy system increases, resulting in a lower success rate.

However, the success probabilities are much lower than expected. For the load balancing proxy with five redirects, we expected more then 40% of successful redirects and achieved only 20%.

This is because in the real world, a DoS attack has two effects: A fraction of requests is not answered at all, but all requests are delayed due to the high load on the systems. In this special case, the Good Client tried to send 600 requests during the experiment, but was limited to a maximum of 10 parallel requests. Only after a reply was received, a new request could be sent.

While in the unloaded case, 600 requests could be handled easily by the system, this was no longer the case under the DoS attack. Therefore we have to distinguish two categories of failed requests:

- Requests that were sent but remained unanswered. These are plotted in Figure 9.9 (c). Different than in Section 7.2.3, we actually observed lost requests in this scenario as the attacked proxy was completely unresponsive.
- Requests that could not be sent during the experiment timeframe, because of delays while answering the previous requests. Those are plotted in Figure 9.9 (d).

The extra delays for successful requests were not directly noticeable to a user, in all cases they were between 120 and 260 milliseconds. Only during the direct attack without proxy defense, the web server's response time increased to 3.6 seconds. Lost requests were detected through a timeout on the client-side. Therefore, each lost request blocked one of the 10 sender-threads for 10 seconds.

Our model only captures the losses of requests and not the timing aspects. When looking only at the lost requests, we can see that the approximation is already much better, even though it still overestimates the effectiveness of the *load-balancing proxy 2*.

It is also interesting to see in Figure 9.9 (c) that the Apache webserver answers a large percentage of the actually sent requests — the main damage is done by delaying requests.

Concluding we can say, that the model used in the previous sections captures the drop behavior of the investigated situations well. Timing is not considered in the model. During a DoS attack, the performance of all considered proxy systems will be negatively affected by extra delays.

Our investigations support the decision not to include timing effects in the model. We have these timing issues as we send batches of requests using a limited number of sender threats. For a user who sends a single request, the effect would have been either a working website (with no noticeable delay) or a timeout.

9.6.1 Practical Aspects

There are a number of practical aspects to be considered when employing a scheme like the one we suggested.

- The server will get requests from only very few IP adresses. If other DoS protection mechanisms had been activated before and are accidentally still active, the proxies could be detected as unusual high-load clients and therefore blocked.

  Further, user identification on the server might take IP addresses as one criterion to distinguish different users (besides cookies, HTTP parameters, ...). Especially the IP address could be used as a negative criterion as it is normally unlikely that the same user sends requests from different IPs. Such interactions have to be considered and solved either by using only reliable information (i.e. from cookies) for distinguishing users or by employing a scheme that determines the last proxy based on the user's real IP address, so the same user will always get the same last hop proxy.

- The proposed scheme will not work if the number of redirects that the client can follow is exceeded. RFC 2068 [52] sets the maximum number of redirects for an HTTP request to 5. RFC 2616 [51] removes this limitation but still mentions that such a limitation existed in the past and that the user should be aware of that. Today common web browsers like Firefox follow 20 redirects.

  When activating this scheme for a web server, it should be considered that the server itself may also use redirects (i.e. when content has changed its location on the server). The last proxy will have to handle such a redirect correctly — either by following the redirect himself and retrieving the right document for the client, or by forwarding the redirect message to the client and being ready to process the client's renewed request.

- According to [51], an HTTP client client has to show a warning to the user when processing redirects of requests that contain HTTP POST information. As a workaround it appears to be the best solution to re-encode POST information as GET parameters during the redirects. The last proxy will receive the encoded GET parameters and can rewrite them as POST parameters for the server. This workaround is not fully compatible due to length restrictions for GET compared to POST parameters.
- As shown in Figure 9.1, a DNS server is used to initially contact the proxies. When a load-balancing scheme is employed, the DNS server should also be kept informed about the set of proxies that do currently accept requests.

  While the proxies provide redundancy, a single DNS server remains a single point of failure which is required for provisioning of the service. Therefore appropriate redundancy is also required here. See Chapter 7 for a discussion of DoS weaknesses in networks caused by service dependencies.
- The scheme might also be used as a part of a larger DoS defense system. The proxies might assign an authentication score to each client, based on different ways to prove the client's legitimacy. So the client might solve computational puzzles if he has JavaScript enabled and perform redirects as proof-of-work if not. This scoring idea might even be combined with captchas. A system that can combine different legitimacy tests has been presented in the related work [16].

9.7 Conclusions

We have shown a proxy-based DoS defense system that purely relies on HTTP redirects.

If JavaScript can be used, a proxy that sends computational puzzles has the advantage that the client can be asked to do more work while the proxy can easily verify the solution. In our solution, the proxy has to invest more resources than the client, due to cryptographic operations (which are cheap, however, due to the limitation on symmetric cryptography).

Our system is compatible with HTTP clients that do not support scripting, i.e. search engine bots. It allows to defend against DoS attacks if the defender has sufficient proxies available.

As already mentioned in Section 9.3.2, the system is meant for dynamic content that has to be generated by the server for each client individually. Static content should be cached by the proxies.

Key findings in Chapter 9:

- **The designed proxy system is effective if the defender has enough resources. However if the use of JavaScript is possible, computational puzzles should be preferred.**
- **Load-balancing acts on information from the past. Therefore it has to be used carefully, a smart attacker can exploit its properties.**

- **The DoS model used in Section 9.5 does not include timing effects. Real-life experiments support the simulation results, but depending on the situation, considering increased delays due to the DoS attack might be important.**

9.8 Acknowledgements

The author would like to thank Wolfgang Zejda who developed the initial proxy and had the idea of how to use the *pcode* parameter for replay protection. Johanna Eicher developed the real-world implementation of the *load-balancing proxy 2*.

Discussions with Marc-Oliver Pahl were very helpful while developing and implementing the ideas presented here.

10. Mobile Worms

Mobile phones are becoming more and more powerful. With fast processors, the ability to install and run a huge amount of client applications and permanent Internet connectivity, today's smartphones have become serious platforms for work and gaming.

Many mobile phones have some form of short-range communications besides the actual cell phone functionality. This is usually Bluetooth, but WLAN is also not uncommon. For the future, Near Field Communication is planned, mainly to support mobile payment applications.

There have been a number of worms for mobile devices, spreading by Bluetooth like Cabir or MMS like Beselo.A, however, no serious outbreak has been observed so far. There is hope that the problem of mobile worms will not become as bad as PC worms, since mobile phones are a relatively closed platform. However phones based on Symbian, Android or Windows Mobile allow the installation of software which has not been tested by the operator. Further it is not unlikely that worms can spread via security holes, as the operating systems of mobile phones are rarely patched.

Cell phones are an interesting target for worms, possible goals of an attacker could be payment systems or spying on the user's location, calls and stored private data.

In this chapter we will describe, what possible harm a mobile worm can do, including a discussion of regional denial of service attacks against the mobile network operator and what defenses look like.

Section 10.1 is about related work. Threats by mobile worms including the novel DoS attack are discussed theoretically in Section 10.2. Section 10.3 will contain simulation results on the feasibility of the attack, Section 10.5 concludes this chapter.

10.1 Related Work

There have been a number of publications on epidemic spreading of viruses for biological diseases but also for computer worms. In the latter case, the authors either use simulations [53] or mathematical models of epidemic spreading [54]. Yan et. al. [55] observed that the mobility model plays an important role for mobile worm propagation. Su et. al. [56] did experiments and simulations, showing that large-scale infections with Bluetooth worms are in fact possible.

So far, a few mobile worms like Cabir[1] and Comwarrior[2] have been observed in the wild. However, none of them was able to spread widely. It appears that the worms were not infectious enough, as they were only able to spread to discoverable bluetooth cellphones with specific operating system versions and even required user interaction. Since mobile phones are becoming more and more powerful and since there is no awareness among the users that security updates might be necessary for their phone's operating systems, the possibility of a large-scale infection is still present.

Security research regarding cellular networks mostly focuses on encryption and authentication issues, but also DoS attack scenarios using signaling messages have been investigated in the past ([57]). The attack we describe here works with user data, however the same effects would also be achievable by causing signaling load on the air-interface signaling channels.

Today, worms on PCs form highly organized botnets. This kind of organization has not yet been observed with mobile worms.

10.2 Threat Analysis

There are plenty of possibilities how a mobile phone can become infected with malware: short-range communications like bluetooth, SMS/MMS, drive-by infection on websites or shipping malware as a Trojan in some seemingly useful software. In this section we will explore possible dangers by mobile malware, without looking into the infection process.

Section 10.2.1 briefly describes general incentives for an attacker to write malware for cell phones. The novel denial-of-service attack will be the subject of the following Section 10.2.2.

10.2.1 Direct Threats for the User

Mobile phones are perceived to be safer than PCs. While most users know that PCs can be infected with viruses, mobile malware has been so rare until now that most people are not aware of it yet. For an author of mobile malware, there are several possible incentives:

- Users increasingly use their cell phones for email and processing other documents. Malware can have the goal of harvesting passwords and other user-specific information.
- If desired, mobile phones allow new ways of spying on people. Not only is it possible to get audio and video from the phone, also tracking the location of the user is possible.

[1]`http://www.f-secure.com/v-descs/cabir.shtml`
[2]`http://www.f-secure.com/v-descs/commwarrior.shtml`

- Mobile phones are used for payment services. Especially banks are sending one-time passwords for PC-based online-banking to cellular phones. Tampering with payment systems could bring a direct financial benefit for the author of the malware.
- Mobile malware could call expensive service numbers or use other paid services.

Further, a mobile phone could also be made unusable by malware, i.e. by changing connection settings (which would cut the phone off from the network) or by automatically launching an application on startup which immediately crashes the phone. Such a destructive behaviour was common to viruses in the past, however it is rarely observed today as it limits spreading and does not give the attacker a benefit.

Defense against such malware is possible by controlling the applications on the device. This would not necessarily mean a complete walled-garden philosophy, the network operator could for example force the users to run a virus-scanner.

10.2.2 DoS Attacks on the Infrastructure

Denial-of-service scenarios against the network operator's core network and Voice-over-IP infrastructure in general have already been discussed in the past ([57]). In this section we focus on possible flooding DoS attacks against individual cells of the radio access network.

An attacker could perform any kind of jamming attack if he had direct access to the device's radio interface. In this section we will make the more realistic assumption that any user-installed software on a mobile phone is bound to the implemented MAC protocols for the cellular as well as the short-range interfaces.

Generally, cellular networks are subject to admission control and apply QoS mechanisms, so they are more robust against DoS attacks than an unmanaged Ethernet or WLAN. A single device is unable to flood a cell, as its share of the medium would be limited by air-interface scheduling.

When multiple infected devices are present in the same cell, an attack is possible. Again air interface scheduling of the operator strongly determines the effectiveness of such attacks. The authors of [58] investigated the capacity of cells in different operator networks and their reaction to high-load conditions. The results suggest that 20-80 voice calls are required to deplete the resources of a UMTS cell. The authors also found out that there is an operator-specific guaranteed minimum bandwidth for each data call and that not all operators prioritize voice higher than data. Highly-loaded cells tend to show strange behavior, including the possibility that all active connections are dropped. Some network operators also offer video calls which are given a relatively high priority while using 3-4 times more bandwidth than voice calls. These results suggest that it is generally possible to DoS an UMTS cell with less than 50 terminals when choosing a combined attack pattern of voice, data and possibly video.

In the following we investigate the threat by a hypothetical malware which targets a network operator's radio cells. There is no point in attacking by blindly sending traffic all the time, this would only make the user and possibly the operator aware of the malware's presence (i.e. by reduced battery lifetime, higher invoices). Usually the volume of data traffic that a user can send per month is also limited by so-called “fair-usage policies”. Therefore the core idea is that the malware only attacks when it knows that a sufficient number of fellow attackers are nearby, so they can collectively DoS the cell.

Such an attack requires coordination between the infected devices, which could be achieved by one of the following approaches:

- **Decentralized Coordination**: When multiple infected devices are close together, they form a (possibly meshed) ad-hoc network, which has the main purpose of counting the number of nearby infected devices. As soon as this number crosses a threshold which allows the devices to DoS the local cell of the mobile network, the devices launch their attack. This method has the advantage that no communication via the cellular network is necessary for the coordination of the attack. Disadvantages are the lack of control by the malware author and that the fact of enough devices being in the cell may not be detected as they are not close enough to form a common ad-hoc network.
- **Central Coordination**: Each malware instance contacts some central coordination point on the Internet (possibly via some proxy to disguise the communication) and transmits its cell ID. The central botmaster can see the distribution of mobile phones and chose to attack cells with enough infected devices. The disadvantage of this approach is the usage of the mobile network for coordination, which may be detected by the operator.
- **Hybrid Coordination**: Bots coordinate themselves via a ad-hoc networks, but only one representative of each ad-hoc network keeps contact with the botmaster. This would still allow for central control, while the amount of revealing signaling traffic is reduced.

In the following section, these approaches for controlling a mobile botnet are simulated.

10.3 Simulations

We have estimated the potential harm that such a worm can do by simulating its behavior for one city. The simulated worm targets one mobile operator network in Munich. This data can be used to estimate what damage a global mobile worm — one that attacks all cells of all network operators — would be able to do. On the other hand an attacker might even want to restrict his worm to a limited area or to cells of a specific operator (compare to the Stuxnet worm [59] which apparently also had a very specific target).

Our simulation is based on the following real data:

- For the population density of different districts, real census data from 2009 was used.[3]
- For the cellular network, the real cell locations of one major mobile operator with a market share of approximately 1/3 were available. The data consists of 2402 GSM and UMTS cell IDs together with the postal address of the building that each base station is located at.

[3] `http://www.muenchen.de/Rathaus/dir/statistik/veroeffentl/39378/index.html`

- The number of distinct postal addresses is only 480. This is because usually **(a)** base stations use three sectoral antennas and **(b)** because it is usual that for example a GSM and a UMTS cell are placed at the same location. The 480 addresses were translated to geo-coordinates using the Google Maps API[4].

- We assume that 100% of the population carry a mobile phone. This is conservative, the real ratio is somewhere between 110% and 120% today. Scaled to the population of Munich[5], which is 1,330,440, we can approximate the total number of mobile phones in the city that belong to our network operator to be 439,000.

Unfortunately there are no good sources on the distribution of certain mobile phone brands/models (only statistics regarding sales), which would have been interesting to further evaluate the results.

It was assumed that a single cell is under DoS when more than 52 worms are active. For this value we took the call capacity of an UMTS cell in voice calls as determined in [58]. We assume that this underestimates the potential harm that a worm on a mobile device can do, as individual devices can produce a higher load if using video or data-calls. Network operators have the possibility to rate-limit data traffic, therefore we chose this minimal value which corresponds to a data rate of approximately 12 kBit/s.

As noted above, usually multiple cells reside at the same location. We do not consider the angle between the cell and the user, therefore we are not able to realistically represent sectoral antennas. So in case of multiple cells at one location, we just added the capacities. This simplification already has a load-balancing effect, so it under-estimates the effectiveness of the attack. The background-load from legitimate users has been set to 15% of the total network capacity, the legitimate users are placed in the city the same way as the attackers.

The simulation calculates a user distribution and its consequences for the network. There is no mobility model included, only snapshots of the simulated scenario are calculated and evaluated. For each user with an infected mobile phone, the simulation performs the following steps:

- A home location for the user is randomly selected on the map according to the city's population density.

- As users do not stay at home all day, a random displacement vector is added. This vector has a gauss distribution, the expected distance of the user from his home is 1 km.

- If the user's calculated location is outside Munich, the whole location selection is repeated. This makes sure that the target number of users is actually reached.

- The cell with the closest distance to the user is calculated. As we do not know details of the mobile network, this cell is taken as the user's active cell. The potential load that can be caused by the worm is added to the cell.

[4]`http://code.google.com/intl/de-DE/apis/maps/`
[5]`https://www.statistikdaten.bayern.de/`

- As the last step, the simulator searches for already-placed users that are **(a)** in the same cell as the new user and **(b)** within an ad-hoc communication range of 30 meters. A user who fulfills both conditions exists, both are assumed to be in the same ad-hoc network.

We expect the users to be more evenly distributed than real humans would be (the random locations do not tend to cluster as much as humans would), which again under-estimates the attack's success. On the other hand in reality the mobile operator constantly enhances his network according to the observed load distribution (if a cell frequently experiences high load, the network capacity in its vicinity will be extended by adding more cells). As we do not have insight into the operator's capacity planning, we can not consider this. This factor may over-estimate the attack's success, as we assume a high call density based on the population density for certain areas, which may not be the case in reality.

As a performance indicator for the attacker's effectiveness we use the number of successfully attacked cells. Further we are interested in the amount of control traffic that is needed to organize the attack.

10.3.1 Attacker Strategy

As discussed in Section 10.2.2, the attacker has three options of organizing his network. This section explains how these are represented in the simulation.

Generally it can be said that the volume of control traffic is determined by the malware's information dissemination strategy. Updates can be triggered, periodic or use combinations of triggered and periodic transmission. The attacker in this scenario is in a similar position as the mobile network operator in the following Chapter 13, therefore many of the optimization versus accuracy considerations apply here, too. In this section we exclusively want to focus on the effects of clustering, therefore we keep our model simple in all other aspects.

We assume that nodes send location updates once per minute. A location update consists of an IP header, a TCP header and the node's cell ID, a 7 byte number. So the total length is 47 bytes.

In case of *Central Coordination* each device sends its updates directly to the bot master. Therefore the volume of the control traffic is $\frac{47 \cdot 8 \cdot \#infected\_devices}{60} \frac{bit}{s}$ for the city. The botmaster makes the perfect attack decision, he will order his worms to attack exactly those cells where a successful attack is possible at the moment.

With *Decentralized Coordination* the worms work fully autonomously, so the control traffic is $0 \frac{bit}{s}$. The worms organize themselves in an ad-hoc network and will only attack if the size of this network exceeds a threshold. In theory this threshold would be set to the number of devices that are needed for a successful attack. In practice the size of a cell is much bigger than the size of a single ad-hoc network. On one hand it is unlikely that an ad-hoc network of 50 devices is formed when users distribute randomly in the simulation (as mentioned before, real humans tend to "cluster" far more, so the attack can be considered more effective in reality). On the other hand the probability of having multiple ad-hoc networks that can not see each other within a cell is high. For the following experiments the attack threshold was set to 5 devices, a choice that is justified by simulations that exclusively focus on this parameter. This means that often cells will be attacked by groups of worms that

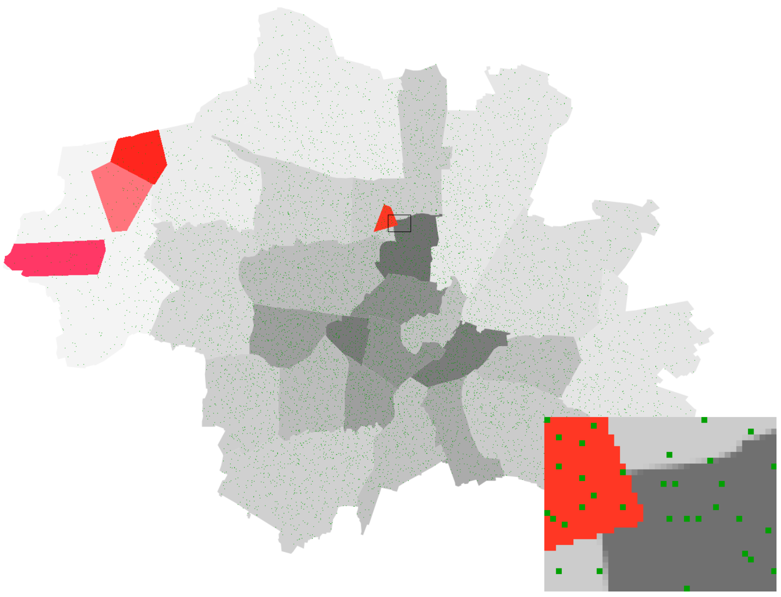


Figure 10.1: Simulation with 10,000 infected users. The magnification shows the marked area near the center of the map.

don't have the critical mass for a full DoS — which is undesirable for the botmaster as it draws attention on the worm without causing real harm.

In case of *Hybrid Coordination*, bots form ad-hoc networks as before. A single representative of each ad-hoc network sends reports to the botmaster. Ad-hoc networks can consist of a single node. The botmaster has the full information and can perfectly organize his attack like with *Central Coordination*, the advantage is that the control traffic is reduced. We assume an update to be one byte larger than with *Central Coordination*, as the number of bots in the ad-hoc network has to be added. The control traffic can therefore be estimated as $\frac{48 \cdot 8 \cdot \#ad\_hoc\_nets}{60} \frac{bit}{s}$.

10.3.2 Simulation Results

To make the simulation method more transparent, Figures 10.1 and 10.2 show a direct simulation output. Here shades of gray show the population density of the districts of Munich. Red areas are overloaded cells, while the specific shade of red varies from cell to cell — this is to make the cells visually distinguishable and has no special meaning. The shape of the cells is due to the simplifying assumption that the closest base station is always responsible for an area.

Tiny green dots are users with infected devices. The internal resolution of the simulation is higher than the resolution of the picture, so a single dot may represent multiple users. In Figure 10.2 users that are part of an ad-hoc network have been colored blue.

In Figure 10.3, simulation results are shown in the form of graphs. In Figure 10.3 (a) the average attack success is shown. Without an attack, an average rate of 2.5 of the 480 cells is overloaded (which equals to 0.5%). 20,000 infected users are required to double this rate in case of *central* or *hybrid* coordination. At 108.000 infected users, more than half of the 480 cell locations would be DoSed.

The *decentralized* worm with an attack threshold of 5 devices per ad-hoc network is by far less effective. It needs 56,000 infected devices to double the legitimate rate of

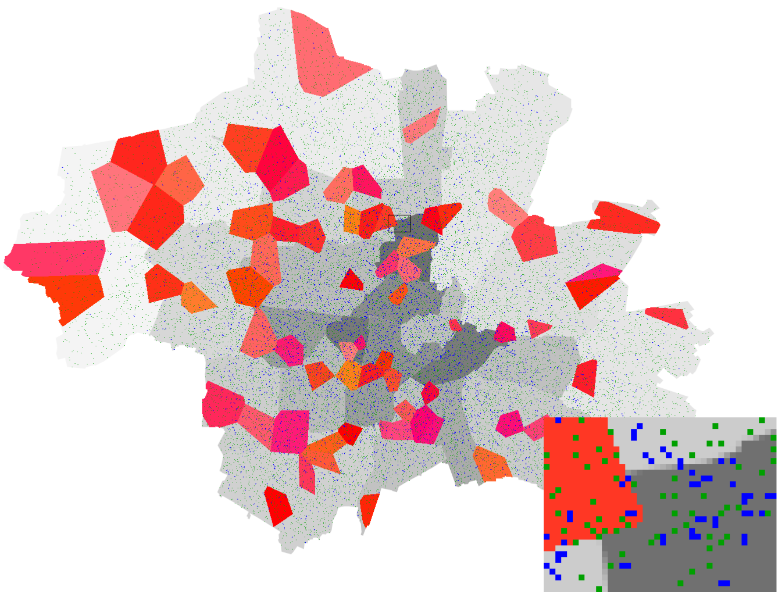


Figure 10.2: Simulation with 30,000 infected users, showing users in ad-hoc networks (with 30m range) as blue dots. The magnification shows the marked area near the center of the map.

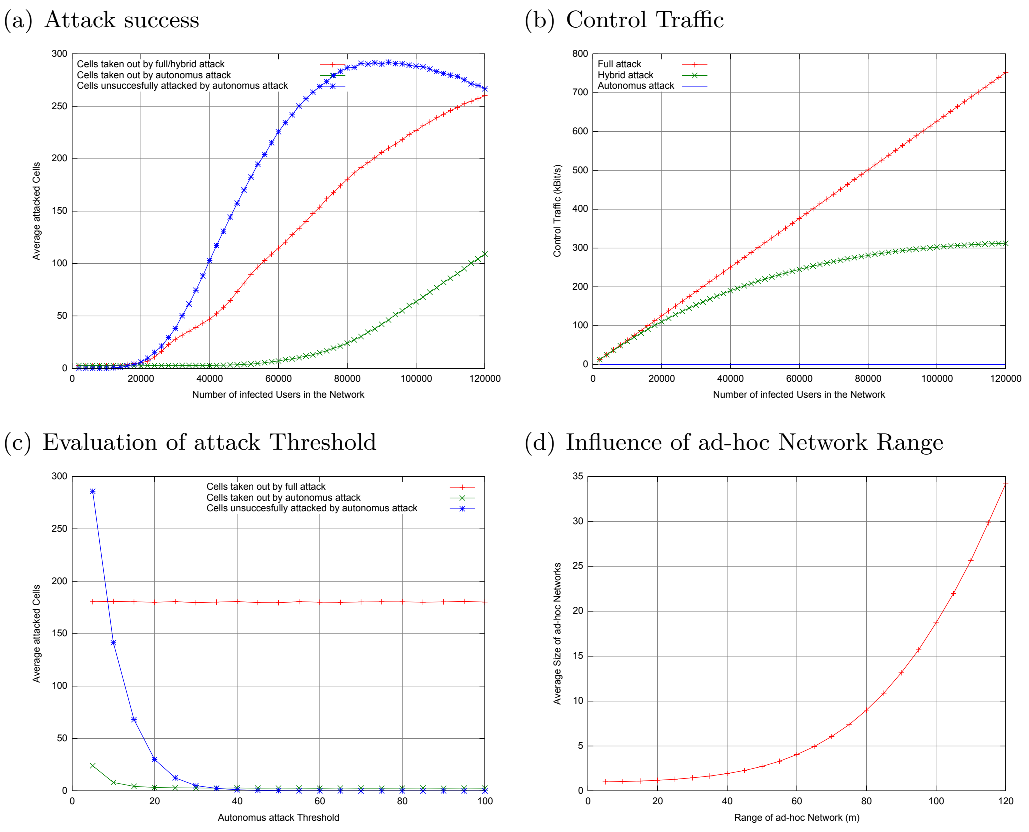


Figure 10.3: Simulation results of the mobile worm simulation. Parameters if not mentioned otherwise: 80,000 infected users, 30 m of ad-hoc network range, autonomous attack threshold 5 devices.

cell overloads to 1%. It also produces lots of unsuccessful attacks, therefore infected devices can probably be identified before causing harm.

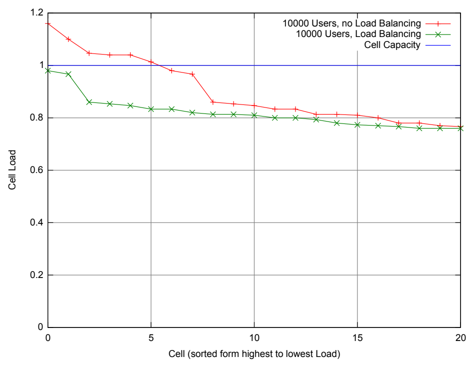
In Figure 10.3 (b) one can see that for this single city, the amount of data traffic produced (depending on the number of infected devices) will be in the order of $1 \frac{Mbit}{s}$, so can still be handled by command and control servers. However, on a global scale, the botmaster would need a significant amount of infrastructure for command and control. The *hybrid* approach reduces the traffic volume especially in case that the worm has spread widely.

Figure 10.3 (c) continues to show that the *decentralized* approach is ineffective. Increasing the network size that triggers an attack disables the worm, as the probability of building a large ad-hoc network is very low. The unsuccessful attacks remain a problem, even when the successful attacks have virtually reached zero. Figure 10.3 (d) shows that an ad-hoc technology with greater range would make this strategy more efficient. At the simulated ad-hoc network range of 30 meters, the average size of such a network is only 1.45 devices. With 100 meters this would increase to about 19 devices.

10.3.3 Load Balancing

Load balancing can help against botnets of this type. Usually a user is in range of multiple cells. Therefore the network operator can decide to move users from an overloaded cell to a less-loaded cell. The bot will notice the changed cell ID, but in the new cell the attack threshold might not be reached, so the bot stops the attack.

(a) Load Balancing with low Load



(b) Load Balancing with varying load

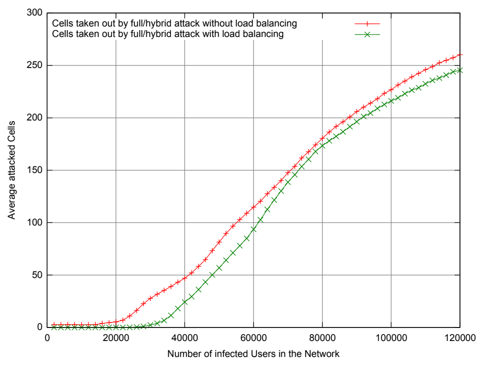


Figure 10.4: Simulation of centrally coordinated worm.

In Figure 10.4, the network operator employs a simple load balancing strategy. If a cell is overloaded, users from this cell are moved to the cell which is the second or third-closest to their location, as long as the load is lower there. After all devices were moved, the botmaster decides again, which cells should be attacked.

Figure 10.4(a) shows the load values for the 20 cells with highest load in case of 10,000 infected users in the system. Load-balancing improves the situation, the number of overloaded cells is reduced from five to zero. In Figure 10.4(b), the general effects of load balancing depending on the number of infected users is shown. Generally it can be said that load balancing helps if the attack is not too strong. If the resources of neighboring cells are exhausted as well, there is no way of improving the situation through load balancing.

Compared to Chapters 8 and 9, it appears like there is no way for the attacker to exploit load balancing. This is because the attacker's possibilities are very limited in this scenario.

10.4 Countermeasures

We see several options to detect and mitigate the threat of this worm.

- In the previous Section 10.3.3 we saw that load balancing can be effective if the attack is not too strong.
- Looking at the communication to the Internet-based botmaster, if existent. However, this is difficult, as already today'S PC-based bots can disguise their traffic, i.e. as normal HTTP communications. Still we suggest that network operators should employ intrusion detection on the mobile device's upstream traffic.
- Detecting short-range coordination traffic, i.e. by placing probes ("Bluetooth Honeypots") at crowded places. This has already been suggested in [56] to detect worm propagation.
- During an attack, the mobile network's QoS mechanisms could be used to keep the mobile network operational. However, this requires the possibility to distinguish the attack traffic from legitimate traffic. In any case voice calls should be given highest priority.
- If the attack started on all devices simultaneously, the operator could use this fact to identify the attacking devices. However, this can be disguised by the attacker by using individual start times for each device.
- If the attack is coordinated locally using an ad-hoc network, the operator could detect the participating hosts by looking at the locations of the nodes in the cell. The attacking nodes are expected to be relatively close together. This might allow blocking the attackers, however, it may also cause some legitimate sessions to be dropped.
- Operators should prevent the creation of mobile botnets by controlling the software on the devices — i.e. a mobile virus scanner could be installed by default.

  Depending on the mobile phone's software platform it might also be possible for the operator to force the installation or un-installation of certain software — so security software might be installed and malware might be removed.

10.5 Conclusions

In this chapter we presented an analysis of the potential danger caused by mobile worms, including a possible DoS attack on mobile operator networks.

It was shown that — even with simulation parameters that probably under-estimate the damage — such a mobile worm is a serious threat. *Centrally* or *hybridly* coordinated, the worm can inflict a lot of damage when infecting enough devices. However these control schemes are not "stealthy", there always is control traffic which can be detected by an intrusion detection system. The completely autonomous version of the worm on the other hand is far less effective when attacking the network.

Load balancing by moving users to different cells is effective as a countermeasure, as long as the attack is not too strong. Similar resource management decisions will be subject of *Part III* of this thesis.

One open question is, whether we really have to expect such attacks. Most of today's worms are used by criminals to earn money. It is unlikely that money can be made from extortion of network operators. However, this attack can also be seen as targeting a geographical region rather than a specific operator, so for example public events could be potential victims — large crowds will also make the attack easier, as more legitimate and more infected users would cluster in a small area. Even if this attack is hypothetical now, the possibility should be considered when designing cellular networks.

Key findings in Chapter 10:

- **The investigated DoS attack on the individual cells of mobile networks is technically feasible and a threat to mobile network operators. With central coordination by a botmaster, the mobile devices will only strike when they can actually cause harm.**
- **The purely decentralized variant of this attack, which is based on ad-hoc networks, has been found to be ineffective. However, this is not a 100% reliable result, as humans tend to cluster more than the randomly distributed users in the simulation. During events which are visited by thousands of people, critical masses of users might still be reached.**
- **It is not clear, what the incentives of an attacker who builds such a worm would be. However, the same can be said about the Stuxnet worm ([59]).**
- **One possible countermeasure is load balancing by moving users to neighboring cells. This improves the situation as long as those cells still have free capacities.**
- **The best defense is the prevention of malware-spreading on mobile phones. This will be harder as these devices become more and more open.**

10.6 Acknowledgements

The author would like to thank Andreas Klenk and Alexander Klein for fruitful discussions.

Part III

Metering and Resource Management in Future Mobile Networks

11. Overview

Future mobile networks should be able to offer seamless handovers between different access technologies. As the management of heterogeneous networks is a major topic in current research, a new term named *Always Best Connected* (ABC) has evolved in recent years [60]. It requires both sides in mobile communication to be considered: On the one hand, users would like to be connected to “best” available network in terms of e.g. signal quality and bandwidth. On the other hand the network provider wants to share the available resources between millions of users in a fair or privilege-based fashion.

Optimal handover decisions between different access networks (so-called heterogeneous handovers) have to take various types of information into account, for example radio conditions and the load of possible target cells. However, collecting this information and transporting it to mobility management decision engines is costly in terms of bandwidth. The benefit on the other side is a more efficient use of network resources, a better user experience and — as we saw in Section 10.3.3 — even improved availability during attacks.

In Section 11.1 the problem statement for Part III of this thesis will be given. Section 11.2 analyzes the current situation in mobile networks, Section 11.3 discusses what data is actually required for heterogeneous handover decisions.

In Chapter 12 we will introduce the **Generic Metering Infrastructure (GMI)**, a modified Publish/Subscribe system for collecting data in mobile networks. Chapter 13 analyzes the trade-off between the quality of resource management decisions and the volume of collected management data when using the GMI.

11.1 Problem Statement

State of the art mobile devices support multiple access technologies and the handover between them. Generally, either the mobile devices themselves or their users decide which access technology to choose. But a mobile-driven handover decision is not always desirable. First of all, this kind of network selection poses a burden on the end user, if it involves user interaction. To hide this complexity from users, smart decision functionalities should be provided that do not involve the user directly. Second, each node has to scan for neighboring networks, which consumes a considerable amount of power. To prolong battery run time of mobile devices, radio transceivers that are not needed shall be turned off most of the time. Third, as each mobile node optimizes only its own connection without considering the impact of its decisions on other nodes, the decision leads to potentially suboptimal solutions. If, for example, one mobile node connects to a WLAN access network while having a bad link and slow modulation, this can significantly worsen the connection quality of other nodes that are also connected to the same access point [61]. Instead it would be better if the node connected to another access technology in which it would not degrade the connection quality of adjacent fellow nodes.

A global view on the network including perfect information about all variables would allow for a centralized decision functionality inside the operator's core network. Such a decision engine would be able to take the current state of numerous users and network-side parameters like the utilization of certain base stations into account. Thereby it could provide better decisions than a single user may make, which promises better performance and allows for a network-controlled sharing of resources in heterogeneous networks.

However the central decision engine would need to obtain all this information somehow. Collecting management data is expensive as base stations are spread in the countryside, with costly rented or wireless "backhaul"-links connecting them to the operator's core network. Management and control traffic have to share these links with user data — and as the customer's data is what an operator is paid for, the share of measurement traffic has to be minimized. It is expected that after the introduction of LTE, the backhaul links will be a major bottleneck for years to come.

The goal of the following Chapters 12 and 13 will be, to get enough information for reasonable handover decisions while keeping the amount of measurement data as low as possible. We will propose a system for monitoring the network and estimate the potential costs in terms of backhaul bandwidth as well as the potential gain.

11.2 Analysis

Although there already is a way of monitoring and managing current provider networks, it lacks of real-time capability [62]. In UMTS networks, for example, mechanisms for performance management exist which allow for requesting various information for individual cells. However, UMTS NodeBs are usually connected to the Core Network by links with limited capacity. In this case, for Operation, Administration and Maintenance (OAM) a channel of approximately 64 kBit/s, is available. The load of a specific cell can be requested using these interfaces. However, the usual OAM interface is realized by uploading ASN.1-files via FTP in intervals of 30 minutes for bandwidth reasons. This kind of interface is not suitable for quick mobility decisions, so alternatives need to be explored.

Flat hierarchies in future mobile networks make the process of data collection even more difficult. In LTE networks there will no longer be a node like the UMTS RNC, which already has load and radio data of hundreds of cells, but only evolved nodeBs located much closer to the antennas. With wireless LAN the situation is similar because all measurement data has to be collected at the access points.

Therefore, it is essential to save bandwidth, especially at bottlenecks in the Radio Access Networks (RANs). Redundant transmission of data must be avoided and the number of measurement reports should be kept as low as possible. On the other hand, the central decision making entities need current information to make good decisions, in some situations it is even necessary to retrieve measurement data on the instant (i.e. in a request/reply fashion).

11.2.1 Granularity Periods

Monitoring in 3GPP networks usually does not directly deal with raw data from the network elements, but with derived **key performance indicators (KPI)**.

A network element collects raw data for an interval called **granularity period (GP)**. After the GP is finished, this data is used to calculate the KPIs [62]. With today's network elements like RNCs, the minimum GP is 5 minutes, while 30 minutes are a far more typical value. Of course monitoring with such a granularity barely helps when building a heterogeneous resource management system, near-realtime data is required for this.

For the future we expect the meters to work the same way as they do today, but with shorter granularity periods in the order of minutes or seconds. Each KPI has a specific minimum GP, new data for handover decisions is only available when a GP has ended.

11.3 Requirements

Based on research of our project partners at DAI-Laboratories [63] four exemplary categories of data have been identified that are required for their **Network Resource Management (N-RM)** decisions.

- *Load information* is required for prevention or handling of overload situations. Thus in case of an overloaded cell or access network, UEs can be moved to different networks.
- *Signal quality* gives the N-RM the chance to move users to a different access network, if the radio conditions are insufficient.
- *Mobility and location information* about a user helps the N-RM to estimate, which alternative networks or cells are available at the user's location.
- *Perceived Quality of Service (QoS)* can be a general indicator that tells the N-RM about a customer whose service quality is degrading. The reason for this will usually be either network overload, an inappropriate access selection or bad signal quality - or a combination of these factors.

Having identified these categories of information we will focus on the question of how the data should be reported to client applications of the GMI (e.g. the Network Resource Management). We distinguish three types of information delivery mechanisms that address different requirements to serve a client's needs:

- *Periodic reports* keep the client constantly informed about the network's state. By monitoring certain measurements the client may be able to act proactively on changing conditions before critical situations occur.
- If nevertheless a critical situation occurs, reports should be *triggered* immediately when the data is available, so the client can react as quickly as possible.
- Additionally, it should be possible for a client to get direct access to a value in a request/reply fashion. Unlike the former methods that require a previous announcement of interest in measured data, such a request is only sent *once* and is answered immediately.

The design of a monitoring system based on these requirements is presented in the following Chapter 12. Simulation results of heterogeneous handovers using the proposed system are shown in Chapter 13.

12. GMI — Concepts and Implementation

To get an efficient view on the state of the network, the **Generic Metering Infrastructure (GMI)** [64] was designed, a publish/subscribe system for collecting and distributing measurement data in an operator's network. It uses various techniques to increase efficiency of data collection, i.e. by building distribution trees, by caching data and by data compression. The GMI is intended for all kinds of management applications, i.e. for fault management, security management, and resource management. This thesis concentrates on the usage of GMI for making handover decisions.

The *Generic Metering Infrastructure* (GMI) is able to provide decision making entities with the desired information and brings the following benefits:

- It can significantly reduce the number of signaling messages by generating optimized information delivery paths and combining multiple aggregation and compression techniques (aggregation of triggers, multicast distribution trees, caching).
- Towards the decision-making entities, the GMI offers an interface that generalizes the configuration of measurement tasks and provides generic API for obtaining measured information.
- It offers an information collection and delivery service that may serve clients in soft real-time and enable faster and more precise decision making.

This chapter is organized as follows. Section 12.1 discusses related work. The description of the GMI concept and design follows in Section 12.2. The components that make up the Generic Metering Infrastructure were implemented, Section 12.3 focuses on this subject. A preliminary evaluation is given in Section 12.4, while more extensive simulations are subject of the following Chapter 13.

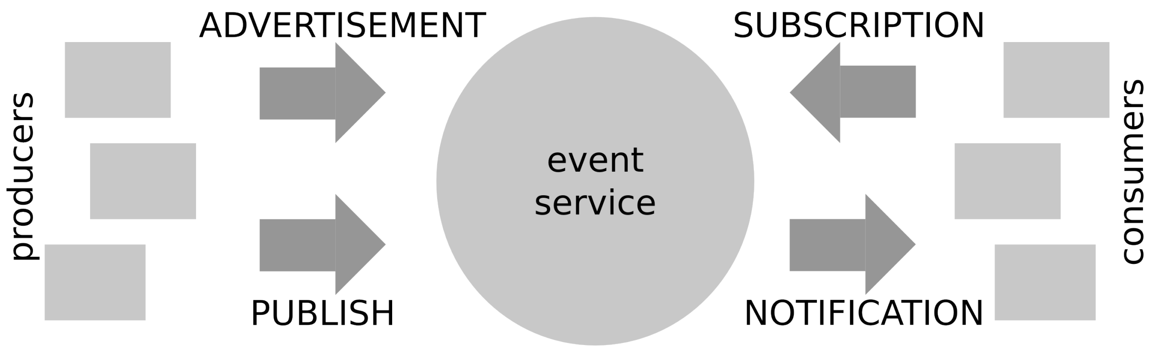


Figure 12.1: Basic concept of an Event Service.

12.1 Related Work

The Publish/Subscribe paradigm is an event-driven way of information dissemination. Publish/Subscribe Systems decouple senders and receivers in two dimensions: Instead of polling the source of information regularly, an interested party registers for events only once (decoupling of time). There might be more than one interested party for some information and the source of the information may not want to or not even be able to send a copy of the information to each recipient. Again it would be desirable to publish the information only once for all recipients (decoupling of space). This concept leads to highly asynchronous communication.

Figure 12.1 illustrates the basic concept of P/S-Systems. As can be observed the interaction is entirely information-driven, thus the source of information and the receiver of information are unaware of each other. Objects of interest or *producers* may *advertise* events to the event-broker system in form of a topic or type that specifies the sort of information they may *publish*. On the other hand, interested parties or *consumers* may *subscribe* at the P/S-System for certain sorts of information they're interested in. As soon as a source of information *publishes* some "news" the event-broker-system starts to dispatch this message and *notifies* all interested parties of the occurred event.

As not all published information is relevant to each of the consumers, there must be a way to reduce the amount of information that will be delivered to a single client. After a consumer submitted a subscription, the event broker is in charge of deciding whether a notification is of interest for a consumer or not. This issue is solved by the introduction of *filters*. A filter is a boolean function that can be applied to all notifications and evaluates to *true*, if the consumer is interested in the notification, or *false*, if he is not.

The filter model determines the degree of flexibility a P/S-System achieves. Basically there are two categories of approaches for filter models. The first category defines a fixed set of topics. The producer of information may decide under which topic it publishes its information.

Channel-based filters offer a predefined set of topics to publish information. The flexibility for classifying the messages is limited by the amount of existing channels, thereby additional filtering of information on the client's side is often needed. The *subject-based* model is similar, but it organizes notification topics (or subjects) in a tree-structure. Again the object of interest chooses the subject to publish its reports.

A subscribing consumer may specify a single leaf of the tree or an intermediate node. After subscribing to an intermediate node a consumer receives all reports that are published at any leaf of the corresponding sub-tree.

These two approaches are simple and easy to implement, but they hinder changes. If the topic-assignment for a type of notification is changed, both producer and consumer have to switch topics simultaneously, to avoid losing information.

The second category of filter models are *content-based filters*, which enable subscriptions that refer to the actual content of a message ([65, 66]). A producer does not have to categorize its notifications anymore. The event system is responsible for deciding whether the information contained inside the message is relevant for each subscriber or not. Although this approach is more flexible in terms of specifying subscriptions, it is also much more complex to realize and requires the costly evaluation of search patterns on each message.

In [67], a publish/subscribe system for mobile devices is developed, with features like location-based notifications. Our work is orthogonal as we build a publish/subscribe system for the network-side.

In the real world, CORBA[1] and Java[2] provide distributed event notification services.

Twitter[3] can be seen as a publish/subscribe system, in which users write and receive news with channel- or content-based filtering. Similar to the GMI, senders and receivers are not completely unaware of each other, as user IDs are used as channel names.

12.2 Design

Most of today's protocols for network management (for example SNMP [68]) are based on the client-/server-paradigm that relies on a closely time-constrained request-/reply-message architecture. But this approach does not suit well for our field of operation. A basic assumption that has already been stated by T. Bandh [69] is that information is of most interest in critical situations. If e.g. the load in a cell rises to a critical level, an N-RM will surely want to be informed immediately and regularly, but if the level of load reduces to a "normal" level the information is of less importance. This observation led to the design of a modified Publish/Subscribe System (P/S-System) which is described in the following sections. Further information is available in [70].

12.2.1 GMI Event-Service: Overview and Signalling

The GMI adapts and modifies the concept of P/S-Systems. Its event-brokers are called Metering Management and Collection Entities (MMCE).

An overview of the GMI can be seen in Figure 12.2. At the top we have the interested parties, which are called metering clients in our terminology. At the bottom we find the actual meters that produce the measurement data, in mobile networks they could be placed at network nodes like RNCs or WLAN APs.

The GMI itself is split into 3 sublayers. The MMCEs that interface directly with the metering clients are primarily meant to route requests to the correct lower MMCE.

[1]`http://www.omg.org/spec/CORBA/`
[2]`http://www.oracle.com/technetwork/java/index-jsp-142945.html`
[3]`http://twitter.com/`

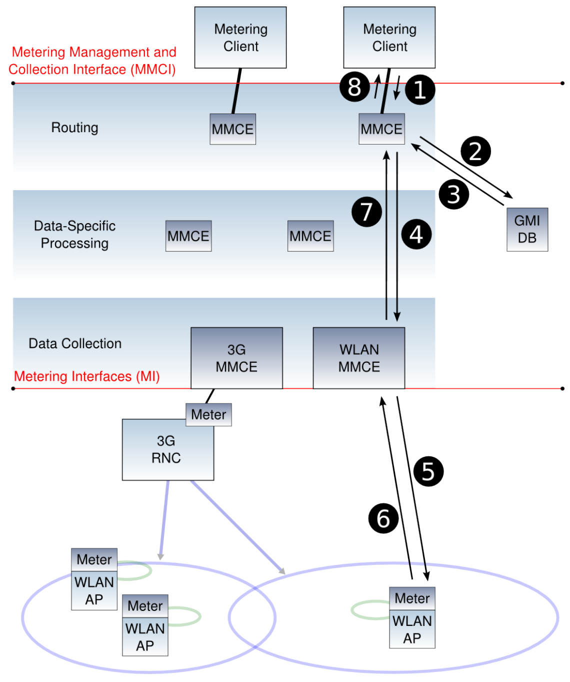


Figure 12.2: GMI signalling overview.

Here it should be noted that MMCEs are logical functions that don't necessarily have to be "physical boxes", an MMCE could also be a process on the Metering Client's machine (analogous to a DNS resolver which is the local component of the DNS service).

Data-specific MMCEs offer additional value-added services to the clients. On this layer we can also build multicast-like distribution trees if multiple clients are interested in the same data.

MMCEs on the data collection layer directly interact with the meters. As the meters may require different protocols for configuration and data delivery, the primary purpose of these MMCEs is translation between GMI messages and the Meter's protocols. Of course one could also develop native GMI meters that do not require this step (again in this case the MMCE could be a software component running on the meter).

Figure 12.2 also shows some example signalling. This is the simplest case without distribution trees or data-specific MMCEs - a single Client requests data from one single Meter.

The Metering Client is interested in some WLAN data and sends a CREATE message to his local MMCE (1). This MMCE performs a lookup in the DNS-like GMI-database (2) to find the source of the requested data. Having received a reply (3), the MMCE forwards the CREATE message to the MMCE assigned for that meter (4). This MMCE is in charge of configuring the meter for this measurement task (5). As soon as new data is available, the Meter sends a report to its assigned MMCE (6). The message is translated into a GMI PUBLISH message which is forwarded to the Client subsequently (7), (8).

12.2.2 Positioning of the MMCEs

Figure 12.3 shows, how the GMI could be deployed in 3GPP's System Architecture Evolution networks. Here a single Network Resource Management (N-RM) instance

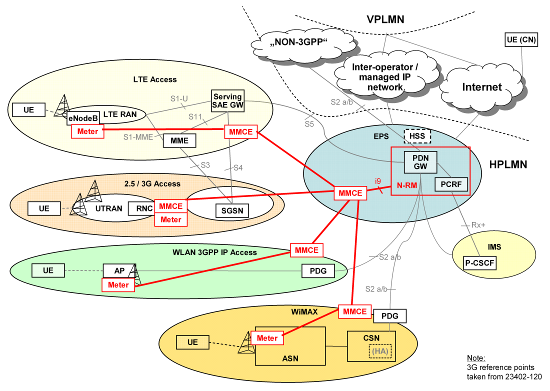


Figure 12.3: Mapping of the GMI to the SAE network architecture.

is the only metering client. In a real network, multiple resource management decision engines and possibly other management systems would obtain their information from the GMI.

All network elements shown in Figure 12.3 produce metering data that is of interest to the clients of the GMI. The MMCEs are placed as close to the meters as possible, but above bandwidth bottlenecks. This is easy to accomplish with UMTS networks, as RNCs are central entities which possess the required information for hundreds of cells. With HSDPA this situation partly changes, as scheduling and radio resource management have been moved to the NodeB, so in this case a Meter on the NodeB is required.

LTE networks have no RNC anymore, the radio resources are controlled by the eNodeB, so a Meter is required there. With WLAN and WiMax, information about the radio links is also available at the actual base stations.

12.2.3 Late Duplication

The MMCEs form an acyclic network of nodes that allows for building up distribution paths for published information that are similar to multicast trees. Thereby the so-called “late-duplication” is applied here. This reduces the bandwidth consumption of the event-system as the messages that have multiple recipients are duplicated as close to the recipient as possible. C. Chalmers [71] investigated the benefit of using multicast trees compared to unicast communication. The general advantage of multicast trees is hard to predict because it depends on multiple factors, such as the breadth and height of the tree and the number of receivers. But it can be observed that the number of messages sent grows logarithmically, if any of the named factors grows linearly. This aspect is especially important in huge networks. By reducing the number of sent duplicate messages to a minimum, the system’s complexity and costs are kept low.

In mobile networks, most of the benefit of late duplication can already be achieved using a very simple strategy. Only one copy of each measurement value should be sent via the backhaul link. As soon as we are in the operator’s core network, data volume is no longer an issue, so from that point we can route individual copies of the data to the receivers.

Such a simple dissemination strategy or the formation of real multicast trees can be organized by the GMI-database. As mentioned before, it is a service similar to DNS, it resolves the ID of a desired measurement value to the address of the responsible MMCE. The GMI-database knows the location of every GMI node, so it can give different nodes different answers to the same request.

For example a client that asks the GMI DB about the location of certain data may be advised to contact *MMCE A* for obtaining it. If *MMCE A* receives the task and does not yet know about the data, it will query the GMI-database again and will receive a different answer — the address of an MMCE which is closer to the data source.

12.2.4 Addressing

The GMI is a modified Publish/Subscribe System. The major difference to other Publish/Subscribe systems lies in the amount of decoupling of senders and receivers. The decoupling in space would prevent the metering clients from directly assigning metering tasks to specific meters. This aspect has been overcome by introducing a scheme that combines the addressing of meters and the filter model of the GMI's event service.

This approach introduces a DNS-addressing scheme that is based on the 3GPP TS TS23.003 [72] (Annex C and D). This standard proposes DNS-like addressing of network functions such as a Serving GPRS Support Node (SGSN). An SGSN can be addressed by appending its name and identifier (e.g.: 1B34) to an operator's top-level domain:

```
sgsn1B34.<mobile national code>.<mobile country code>.3gppnetwork.org
```

3GPP only defines DNS-names for those network nodes that are addressable by IP. But as we want to be able to address meters like UMTS NodeBs that are not capable of IP, we propose an easy extension of the naming scheme to these entities.

The introduction of a new MMCE-domain below the operators top-level domain spans a new overlay network of Metering Management and Collection Entities where each meter forms its own DNS-domain within the mmce-domain. That way a self-explaining addressing scheme can be deployed, for example data regarding an UMTS NodeB could be found under the address

```
nodeB0123.mmce
.mnc123.mcc123.3gppnetwork.org
```

Obviously this scheme introduces an intended indirection. The DNS-names do not refer to the meters themselves but to their correspondent MMCEs. They are responsible for administering metering tasks and forwarding measurement reports.

12.2.5 GMI Subject-Tree

The GMI uses the subject-based filter model. Because the previously defined domain-names already imply a hierarchical structure, they can directly be mapped into a subject-tree. This tree does not only allow addressing of meters but it also contains addresses for the data that is measured there. Each measurement forms a new sub-domain within its meter's domain. That way, a measurement can be addressed by appending its name to the address of the meter.

For instance the topic *FailOutintraNodeB* (number of failed outgoing intra-NodeB hard handovers) can be measured at each NodeB [73]. Because there can be multiple reasons for a handover-failure this topic is split up into several reasons or sub-measurements. Among others these can be: *NoReply, ProtocolError, RadioLinkFailure* or *sum.* The resulting name of the measurement *NoReply* would be:

```
NoReply.FailOutintraNodeB.nodeB0123
.mmce.mnc123.mcc123.3gppnetwork.org
```

It shall be understood that the other parameters can be addressed accordingly.

These capabilities can be advertised on start-up of the network to the attached MMCE of a meter. Thereby each MMCE has to cope only with the information of its attached meter and doesn't have to be aware of the information that can be found on other Metering Management and Collection Entities distributed in the network.

12.2.6 Measurements and Events

As already mentioned in section 11.2, a basic requirement of our architecture is the support of a flexible set of measurement tasks.

To emphasise that a subscription in the GMI's sense differs from the classic sense of a P/S-System, we replaced the *SUBSCRIBE* message with a *CREATE* message. Its basic functionality remains the same (declare the interest in a certain type of information). But a CREATE-message may also cause new measurement tasks to be created at the meter.

Periodic measurements: A metering client can subscribe to periodic metering tasks. In this case the subscriber specifies a desired **report period (RP)** (usually a multiple of the *granularity period* introduced in Section 11.2.1) and the measurement value it wants to stay informed about. If there already is a subscription that matches the desired measurement, the sender of the *CREATE* message is appended to the already existing list of receivers. If there is no matching subscription, a new metering task will be started. Thereby the GMI ensures that only one periodic measurement task with the given report period is active at a time.

Triggers: It is also possible to set triggers for measurements. Such a subscription sets one or multiple thresholds for a metered value. If the value rises above or falls below the given threshold the metering client is informed immediately. Trigger subscriptions contain a hysteresis parameter to make sure that a value which oscillates around the threshold does not cause an unnecessarily large amount of messages. A trigger notification consists of two values: the former value and the currently measured value that caused at least one trigger to fire. This enables an implicit aggregation of triggered measurement reports. If a single new value causes multiple triggers to fire, only one message is sent. Each intermediate MMCE in the distribution tree can decide which subscribers of triggers have to be informed. This enables “late duplication” according to the classic P/S-scheme: Messages that are of interest for multiple recipients are duplicated on their way to the destinations as late as possible to avoid redundant message transmissions over the same links. Additionally, a client may subscribe to classic events like handovers and connection losses. These events are not associated with a numerical measurement value inside the meter, but stand on their own.

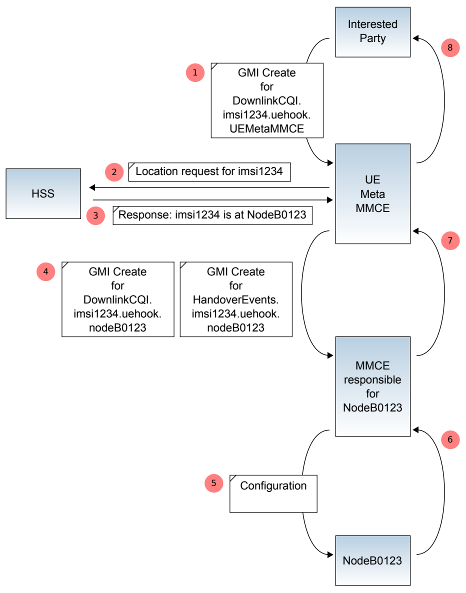

Interested Party
1
GMI Create for DownlinkCQI. imsi1234.uehook. UEMetaMMCE
8
2
Location request for imsi1234
HSS
UE Meta MMCE
3
Response: imsi1234 is at NodeB0123
7
4
GMI Create for DownlinkCQI. imsi1234.uehook. nodeB0123
GMI Create for HandoverEvents. imsi1234.uehook. nodeB0123
MMCE responsible for NodeB0123
6
5
Configuration
NodeB0123

Figure 12.4: Concept of a data-specific MMCE that hides UE-mobility from the clients.

Request/Reply: The last type of reporting is an immediate response to a request of a metering client for a certain value of data. This notification is not an event in the classic sense of a P/S-System. In this case the metering client simply sends a request for the data (which is a message similar to a subscription) and receives a reply containing the value (which is handled like a notification). In this case no aggregation is possible as the message is only sent to a single client. However caching of values can limit the number of requests to the meters, if the cached information is still current enough.

12.2.7 Tracking mobile Sources of Data

As mentioned before, subject-trees do have their drawbacks, especially when the source of certain information changes its location. For example, an N-RM application may want to be kept informed about the signal quality of a single user. But the source of information may change when the user switches to a different base station. If the metering client is unaware of that, it will not receive updates anymore.

This means that the proposed approach has to be extended to meet the requirement of keeping track of changing sources for the same data. The introduction of "*hooks*" is meant to address this problem. Event producers with dynamically changing subjects of information may advertise *hooks* to announce predefined sorts of information, that contain variable content.

A hook is defined by a template and its available instances. The template defines a sub-tree structure containing measurements that can be found at each instance. Each meter that can provide dynamically changing data must use hooks to announce its changing capabilities. An MMCE that is attached to such a meter advertises a template for its hook and its current instances instead of only specifying a static configuration. An example for such a hook is

```
DownlinkCQI.imsi1234.uehook.nodeB0123
 .mmce.mnc123.mcc123.3gppnetwork.org
```

A so-called "data specific MMCE" (DS-MMCE) can be used to hide this mobility. Such an MMCE would offer data about all of the operator's UEs or a subset thereof. When first receiving a request about a specific UE (step 1 in Figure 12.4), it needs to look up the UE's location at the HSS (step 2 and 3). It subscribes to the data at the responsible low-level MMCE, and sends an additional subscription to get notified about handovers (4). It receives the measurement data (7) and forwards it to the client (8). In case of a handover, it would be notified about this event, delete the old subscriptions and subscribe to the UE's new location. Therefore this DS-MMCE makes the data available to other MMCEs and clients at a constant location.

12.2.8 Generic Measurements

Another feature of our system are so-called "generic measurement tasks". Meters can contain plug-in modules for specific tasks, i.e. flow-based QoS measurements. A configuration for such a plug-in may be sent to a meter using normal GMI mechanisms, which means that it is encapsulated in a GMI-subscription. At the meter, the configuration is forwarded to the plug-in. The results of the measurements are assigned to an identifier (i.e. a flow ID) and published in the P/S tree at a special hook.

```
AvgDelay.flow0123456.qosplugin.ggsn0123
  .mmce.mnc123.mcc123.3gppnetwork.org
```

This is useful when a metering task requires configuration that is too complex to be encoded in a GMI address.

12.2.9 Interface towards the Clients

The GMI is a service that acts as a middleware between metering clients and the meters. A metering client is expected to connect to only one MMCE that serves as its access point to the GMI. This MMCE is assigned by the network operator.

The GMI provides an abstraction layer that allows a metering client to create measurement tasks for every meter within the network the same way. The message format for different meters (e.g. a WLAN access point and an SGSN) has the same structure although the actual configuration of measurement tasks at these meters may be very different since vendor and implementation-specific aspects often have to be considered. Here the lowest MMCE, which directly interfaces with the meters, is in charge of translating the requests according to the meters' specification. Thereby a metering client does not have to worry about device-specific aspects of different meters.

12.3 Implementation

The key components of the envisioned system have been implemented in our laboratory. As the GMI is middleware, it needs data sources and metering clients to run. As a data source we have implemented a network emulation application, which allows simulated users to move on a map, connect to different radio cells and start sessions. Each cell maintains a capacity counter. The model is simple, but it allows for homogeneous and heterogeneous handovers and is sufficient for initial tests of the GMI. Figure 12.6 shows a screenshot of our simulator.

```
<gmi-message messageType="CREATE">
  <header>
    <sender ip="127.0.0.1" />
    <receiver ip="127.0.0.1" uri="cell0.mmce.plmn" />
    <messageId>127.0.0.1:0</messageId>
    <timestamp>2007-11-12T15:59:39</timestamp>
  </header>
  <node name="plmn">
    <node name="mmce">
      <node name="cell0">
        <node name="bandwidth">
          <node name="used">
            <periodic repPeriod="3" startAt="2007-11-12T15:59:39"
                                    stopAt="2007-11-13T15:59:39" />
          </node>
        </node>
        <node name="numSessions">
          <periodic repPeriod="7" startAt="2007-11-12T15:59:39"
                                  stopAt="2007-11-13T15:59:39" />
        </node>
      </node>
    </node>
  </node>
</gmi-message>
```

Figure 12.5: A GMI CREATE message.

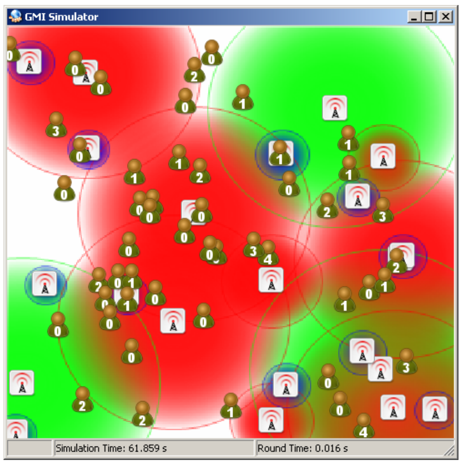


Figure 12.6: Screenshot of the Simulator that generates input data for the GMI. Here three different access technologies (UMTS, WLAN, LTE) and 50 mobile users are shown. The number printed on the users are the currently active sessions.

Our MMCE instances maintain the subscriptions in a tree. In our current implementation all messages are XML-based (the choice of XML as the message format will be discussed and evaluated in Section 13.4) and also interpreted as tree-like data structures.

Figure 12.5 shows a GMI CREATE message. The “node”-elements form a subtree of the GMI subject tree, the “periodic”-elements create new periodic measurement jobs for the used bandwidth and the number of users of the given cell. The message is addressed to the responsible MMCE for this cell.

Metered information must be advertised in advance, so the system can add it to the subject tree. New meters and new values in existing meters can be dynamically added and removed at runtime. The MMCEs that receive advertise-messages store routing information as annotations in their local subject-tree datastructure. Each MMCE only holds the routing information that is relevant for its own operation.

Received “PUBLISH” messages are interpreted as a subset of the subject-tree. The forwarding decisions made at each intermediate MMCE are based on an algorithm that traverses the received tree node by node and matches it to the subject-tree. This easily allows to determine the receivers of an event.

We have implemented our concept in the Python programming language. Our MMCEs are individual applications that communicate via TCP sockets. An initial evaluation of the metering possibilities is given in the following Section 12.4.

12.4 Data Accuracy of Periodic Measurements and Triggers

In this initial evaluation, the load of one cell of our simulator is measured. The load value is updated at the meter in intervals of one second. A light-weight metering client sends a subscription for the data and compares the received result with the expected original curve using the $L2$ norm. Basically a subsampling of the original curve is applied, for example each 10th value is transmitted if the metering task is set to “periodic, 10 seconds”.

Besides using simple periodic measurements, we tested a combination of periodic measurements and triggers. The upper hysteresis thresholds of four triggers are set

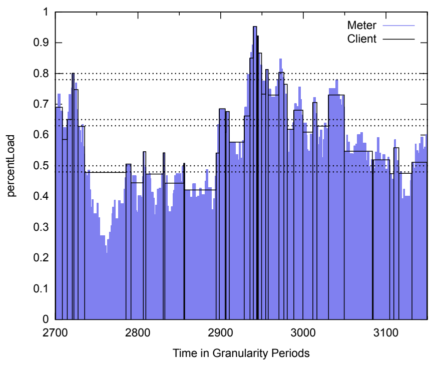


Figure 12.7: Example from our evaluation data set. The filled curve is the original load value that appears in a simulated cell (view of the meter). The black curve is the output of a GMI-measurement job using four triggers (view of the GMI's client). Pairs of dashed horizontal lines indicate the upper- and lower thresholds for the triggers.

to 50%, 65%, 80% and 90% of the cell's total capacity (the lower thresholds are at 48%, 63%, 78% and 88%). Figure 12.7 shows a small part of the data sampled with these triggers. Here the filled curve is the original data while the black lines show what the metering client sees.

During the 10 000 seconds of simulation time, the load in the cell varies, but it is generally above 50%; some peaks even touch the cell's capacity limit.

Figure 12.8 shows the result of the evaluation. Here the accuracy of the measurement ($L2$ distance between measured curve and real curve) is drawn against the number of reports that had to be sent to achieve this accuracy - so values that are closer to the origin are better. With the "periodic" curve, one can see the trade-off between accuracy and report period - more reports produce a higher accuracy. The "both" curve shows that adding triggers increases the accuracy. Without triggers, a report period of 5 seconds (which equals to 2000 messages) is needed to achieve an accuracy of 0.0026 in the $L2$ norm. The same accuracy can be reached by activating the triggers as described above and setting the periodic report period to 35 seconds - and with this setting only 1470 messages are necessary.

In our tests, the size of the GMI-XML PUBLISH messages was approximately 370 bytes per message on application layer. However, XML is only used in our implementation and not conceptually required, the message size can be reduced by a factor of 10 when using a more compact representation like WBXML ([74]). The aspect of data volume is further discussed in Section 13.4.

As expected we see that more measurement values lead to an improved accuracy. However, we also see that the accuracy can be improved by smart metering, the curve using triggers is generally better than the pure periodic curve as triggers are better for capturing changes. It should be noted that the evaluation metric did not even consider the effect that motivated the introduction for triggers — the fact that some information (e.g. the load in an unloaded cell) is simply not interesting.

12.5 Conclusions

In this chapter we have presented a publish/subscribe system for distributing measurement data in future mobile networks.

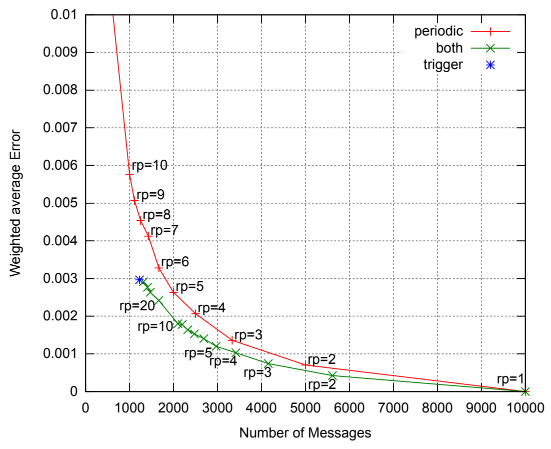


Figure 12.8: Evaluation results: Measurement accuracy versus the number of sent messages.

As a hierarchical naming scheme already existed in mobile networks, these names were used to populate the subject tree of the publish subscribe system. This reduces the “decoupling of space”, however, in an environment with standardized names this is not a huge disadvantage.

In mobile networks, three ways of requesting data are desirable: Periodic reporting, triggered reporting and explicit requests for certain data. We have implemented these, an initial evaluation showed that these flexible report types help to achieve a higher accuracy than purely periodic reporting.

Triggers can be aggregated to a single message by sending the current and the last value of the associated variable. This way each entity that has to process the trigger message can determine by itself, which triggers have fired.

The GMI was designed to transport measurement data that is required for centralized control of heterogeneous handovers. In the following Chapter 13 we evaluate the GMI in this use-case.

In the implementation, the GMI’s message format is based on XML. This choice was mainly made to simplify the implementation in our demonstrator. In a real network, a more compact representation of the data is desirable. This open point of the GMI’s design will be discussed in Section 13.4 in the context of simulation results.

12.6 Acknowledgements

Several people were involved in the design, implementation, evaluation and demonstration of the GMI. A large contribution was made by Andreas Monger who did a lot of conceptual work as well as the initial implementation of the MMCEs for his diploma thesis and later helped to build a demonstrator of GMI components. Mark Schmidt implemented large parts of the simulator and also enhanced the GMI in his diploma thesis. Vladimir Maldybaev also participated in implementing the GMI demo.

For further fruitful discussions I want to thank Tobias Bandh, Christian Hoene, Morten Schläger and Frank-Uwe Andersen. Frank-Uwe also created the picture that was re-used by me as the basis of Figure 12.3.

13. Using the GMI for Resource Management in Future Mobile Networks

The evaluation of the GMI is a difficult task, since the software is middleware. The behavior and the performance of the GMI greatly depend on the underlying meters and — especially — on the subscriptions of the metering clients.

Section 12.4 uses a pre-defined data set and measures the error between the actual and the transmitted data with periodic or triggered sampling. This work was conducted as an initial evaluation of these two methods. However, the evaluation metric does not incorporate the actual incentive for introducing triggers: In many cases we only need information if some measurement value is in a critical region. For example we do not need to monitor the load of cells constantly if it is low, we just have to be informed at once when an overload occurs.

Therefore, we built a simulated environment that mimics the actual use-case of the GMI. Besides the radio network simulation we added a network resource management (N-RM) that bases its decisions on data supplied by the GMI.

After discussing related work in Section 13.1, the complex evaluation scenario is explained in Section 13.2, results are shown in Section 13.3. Section 13.4 translates the results from the simulations into the real world.

13.1 Related Work

In recent years, there has been a lot of research on handovers in heterogeneous networks. The approaches in [60], [75] and [76] leave the decision which network to choose to the mobile terminal. This is a reasonable design concept since the information about signal quality of the surrounding base stations is available there, while with network-centric decision engines this information must be transported to the core network. Transporting this data also introduces undesired delay that leads to potentially imprecise values.

On the other hand a management facility inside the network is able to take global information, e.g. on the load situation into account and therefore is able to make better decisions. Additionally, it can help the mobile terminal to find adjacent networks without forcing it to scan for available access points which would deplete its battery power.

The authors of [77] use a network-centric approach which basically integrates WLAN into an UMTS UTRAN network. Our approach attempts to be more general by abstracting the handover-logic from details of the RANs. The authors of [78] adjust load triggers on the network-side to optimize handover performance. This work is about the actual handover decision process, i.e. in a combined GERAN/UTRAN network, while we primarily focus on data collection and intentionally keep the decision process as simple as possible. In our approach ping-pong effects are mitigated by the score calculation of the local resource management (Algorithm 3).

The authors of [63] use a network-assisted policy-based approach. They're using two decision engines, one of which is located in the core network while the other resides on the mobile terminal. This scenario is also the base of our work, the GMI could be used here to provide the network and RAN-related information which is required by the decision engine on the network side.

A related standard is IEEE 802.21 [79], which specifies an information service, an event service and a command service to support heterogeneous access decisions and therefore consists of several building blocks that were similarly realized for our simulations. However IEEE 802.21 leaves the actual question of data transport open.

13.2 Setup for N-RM Experiments

In this section we describe the setup of our experiments. All components of our simulations can be seen in Figure 13.1, and are described in this section. They were run on a single machine, but as separate applications using TCP/IP communications.

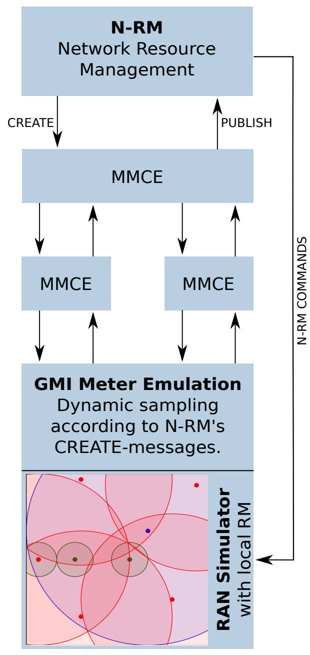


Figure 13.1: Experiment setup including N-RM.

13.2.1 The User and Radio Network Simulator

The simulator is an application that simulates the mobile networks and their users. Several cells belonging to different **radio access networks (RANs)** are placed on a map. The users move around on this map and start and stop sessions. We use an "accelerated real-time" simulation, events in the simulator happen 15 times faster than they would in reality. The simulator is able to accept GMI subscriptions for a large number of parameters regarding cells and users. The minimum granularity period of all meters is one second, which would be 15 seconds in the real system. In the following text we will give the report period in multiples of the granularity period — and not in seconds — to avoid confusion between real time and simulated time.

13.2.1.1 Map and Radio Access Networks:

The map which was used in our simulations is shown in Figure 13.2. There are three different radio access technologies, which differ in the capacity and range of their cells. The range is given in meters while the capacity is defined as the number of simultaneous sessions that a cell supports. Sessions require a fixed amount of bandwidth and there is only one type of sessions. Compared to the cell capacity, the sessions can be regarded as video calls. We also do not distinguish between upstream and downstream traffic.

RAN A is a network of high capacity and high range, it could be realized using real-world technologies like LTE or WiMAX. *RAN B* is a cellular network. The individual cells have a lower capacity, but *RAN B* has coverage everywhere on the map and in most areas there is even an overlap between the cells. One could imagine *RAN B* as being GSM or UMTS. *RAN C* consists of cells with relatively high bandwidth but a very small range — they can be considered to be WLAN hotspots. On our map there is a total of nine cells in an area of 1.44 km^2. We assume freespace radio propagation, users will lose connectivity as soon as the distance to the base station is larger than the range given in Table 13.1.

There is a network-internal resource management built into the simulator, which allows handovers between cells of the same RAN, the handover logic for this case is given in Algorithm 3 which is executed for each user periodically.

Access Technology	Number of Cells	Capacity (simultaneous Sessions)	Range (m)
RAN A	1	31	800
RAN B	5	12	600
RAN C	3	20	120

Table 13.1: Different radio access networks.

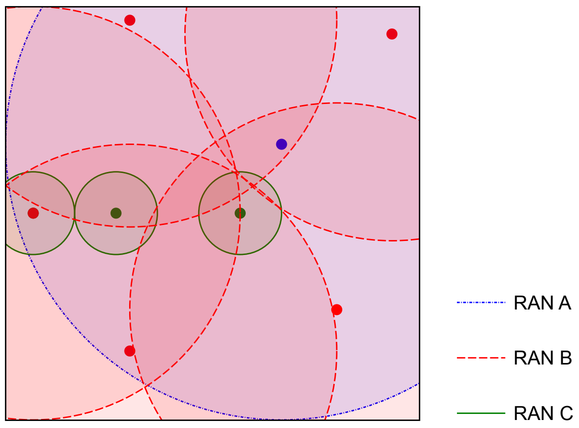


Figure 13.2: The map for the simulations, with nine cells of three different radio access technologies.

13.2.1.2 Users:

The movement model of the simulated users is a modified random waypoint pattern. The users select a new waypoint somewhere on the map with a probability of 40%. With 10% probability they will move to one of the three interesting locations that are covered by the *RAN C* hotspots. In 50% of the cases a user will stay at his current position for an exponentially distributed random time. On average, users move at a "pedestrian" pace of 1 meter per second.

In reality users of mobile devices are inactive most of the time, but for our simulation it makes little sense to model inactive users. To keep our model simple, we have chosen users who start sessions with an exponentially distributed inter-arrival time. Session length is exponentially distributed with the same expectation. By this construction, each user has 0.5 active sessions on average.

The users produce actual load on the air interface by opening sessions, while the GMI produces signalling load which only occurs on the backhaul links between the base stations and the core network. Our assumption here is that there is no direct interaction between these two kinds of traffic, even though in reality they have to share the backhaul link. One primary goal of this paper is to estimate the load on the backhaul link which is caused by the measurements — therefore, this parameter is monitored but not limited.

13.2.2 The MMCEs

As explained in Chapter 12, the MMCEs are the message brokers of our publish/subscribe system. In a real network they would collect data for the N-RM and other applications from various different locations in the network. In our current setup, all data comes from the simulator.

Algorithm 3: Mobile terminal and RAN-internal RM decisions

```
input     : users, cells
constants: $\delta > 0$
foreach $u \in users$ do
    // mobile terminal-side logic.  If N-RM recommended cells, the mobile
       terminal tries to connect there.
    foreach $c \in cells\_recommended\_by\_global\_rm(u)$ do
        // check signal quality
        if $sq(c, u) > th_{acceptable\_sq}$ then
            $u.handoverTo(c)$;
            $continue\_with\_next\_user$;
    // This is the "local RM" logic.  It only makes handovers within cells
       of the same RAN.
    foreach $c \in nearby\_cells\_of\_current\_ran$ do
        // check minimum requirements for load and signal
        if $sq(c, u) > th_{critical\_sq}$ and $load(c) < th_{critical\_load}$ then
            // find best available cell
            $score(c) \leftarrow calculate\_score(load(c), sq(c, u))$;
            if $score(c) > score(former\_best\_cell) + \delta$ then
                $new\_best\_cell \leftarrow c$;
    if $new\_best\_cell$ found then
        $u.handoverTo(new\_best\_cell)$;
    else
        // stay in current cell.
```

Algorithm 4: N-RM decision logic on bad signal quality

```
precondition: u ∈ users, sq(u) < $th_{critical\_sq}$
input       : users, cells
$cells_{recommended} \leftarrow \emptyset$;
foreach $n \in neighbour\_cells(cell(u))$ do
    if $load(n) < th_{acceptable\_load}$ then
        $cells_{recommended} \leftarrow cells_{recommended} + \{n\}$;
sort_by_load($cells_{recommended}$);
$send\_recommendation(u, cells_{recommended})$;
```

Algorithm 5: N-RM decision logic on cell overload

```
precondition: c ∈ cells, load(c) > $th_{critical\_load}$
input       : users, cells, user_list(c)
$cells_{recommended} \leftarrow \emptyset$;
foreach $n \in neighbour\_cells(c)$ do
    if $load(n) < th_{acceptable\_load}$ then
        $cells_{recommended} \leftarrow cells_{recommended} + \{n\}$;
sort_by_load($cells_{recommended}$);
$receivers = random\_subset(users(c))$;
foreach $r \in receivers$ do
    $send\_recommendation(r, cells_{recommended})$;
```

13.2.3 N-RM

The Network Resource Management (N-RM) is our global application for handover management. It gets data from the simulator via GMI after subscribing by sending CREATE messages. Its decisions are passed back to the simulator and influence the mobile terminal's cell selection (see Figure 13.1).

We assume that N-RM does not know the exact location of the user — it only knows the user's location on a "per cell" granularity. Therefore, it can not tell exactly, which cells are available at the user's current position, it only knows which cells overlap.

The mobile terminal initially does not know about cells of other RANs. In real life it would have to scan different frequencies to find alternative cells — and this constant scanning would waste battery power. So in our case, N-RM will give the mobile terminal hints, which cells to search for. In reality these hints would also contain radio parameters. The mobile terminal will only scan for cells of different RANs after receiving such a hint or after losing connectivity with its current cell.

We are testing five different setups:

- *Experiments without N-RM*

  In these runs, the users will stay in their *RAN* as long as they have connectivity. A user who leaves the coverage-area of his RAN will lose his active session, then he will start scanning for other radio access networks and connect to the strongest cell he receives. Therefore there are no inter-RAN handovers without N-RM.

- *Experiments with a purely trigger-based N-RM*

The N-RM subscribes to the load in each cell and the signal quality of each user. The subscriptions are trigger-based, which means that N-RM is notified whenever a threshold-value is crossed.

In case of a user with bad signal quality, N-RM will request load information from surrounding cells in a request/reply fashion. Based on the results it will recommend cells to the user (see Algorithm 4). This recommendation will have an impact on the next run of the local RM (Algorithm 3) inside the simulator.

In case of an overloaded cell, N-RM will again be informed by a trigger and then request load information from the neighbour cells. It will choose a subset of the current users in the overloaded cell and send its recommendations to this subset (see Algorithm 5).

With this basic triggered N-RM, the resource management will only be informed when a critical situation occurs, there is no feedback whether the situation persists despite the countermeasures. As our triggers are defined with hysteresis, N-RM will only take action again when i.e. the load in a cell crosses the upper threshold again after it has crossed the lower threshold.

- *Experiments with an N-RM based on periodic reports*

  N-RM subscribes to load information of each cell and signal quality information of each user on a periodic basis. This means that there is a constant flow of reports which does not change during the simulation. When receiving information, N-RM will check if a parameter is critical. With this measurement strategy N-RM will never request current information when it needs specific data, but it uses the last reported value.

  The basic decision algorithms are still Algorithm 4 and Algorithm 5. These are run periodically whenever new data has arrived.

- *Combination of triggers and periodic reports*

  This variant works with triggers again, but after a trigger has fired and N-RM has taken action, it enters a success control loop. In this state the trigger is turned off and replaced by a periodic measurement job which continuously monitors the critical value.

  With each arrival of a current value, N-RM will check if the system still is in a critical state. If this is the case, it will again request additional data which is needed for the decision and take action accordingly. There is a different (much lower) threshold for cancelling the periodic measurement job and returning to triggered measurements when the situation has been resolved.

- *N-RM with full information*

  For comparison all simulations have also been run using a N-RM which subscribes to periodic reports of all values in the simulator with a report period of 1 GP. This can be seen as the theoretical maximum amount of data that N-RM could possibly subscribe to. The decision algorithms are still the same.

13.2.3.1 Triggers and threshold values

Table 13.2 shows the different threshold values that were used in the simulations. Always the “critical” threshold (0.94 for the cell load, 0.06 for the signal quality) is the one that causes N-RM to take action when being crossed. For the cell load of *RAN B* this concretely means, that N-RM will only send recommendations when

Parameter	Threshold	Value
Load	Hysteresis higher threshold ($th_{critical\_load}$)	0.94
	Hysteresis lower threshold ($th_{acceptable\_load}$)	0.89
	Control loop stop threshold	0.70
Signal Quality	Control loop stop threshold	0.15
	Hysteresis higher treshold ($th_{acceptable\_sq}$)	0.07
	Hysteresis lower threshold ($th_{critical\_sq}$)	0.06

Table 13.2: Threshold values for N-RM.

a cell is operating at full capacity. This is by design, as the load reduction procedure causes a lot of overhead and multiple handovers, so it should not be triggered unnecessarily.

The "acceptable" threshold (0.89 for the cell load) re-activates the load-trigger after it has fired once. This avoids a message storm if the value oscillates around the actual critical threshold.

The third threshold (0.70 for the cell load) is only active when triggers and periodic reports are combined. Crossing this threshold from above causes the reporting to switch back from periodic to triggered.

13.2.4 Simulation Flow

On startup, users will connect to the strongest cell in range. *RAN A* and *RAN C* have no overlapping cells, so there are no horizontal (intra-RAN) handovers. For users of the cellular *RAN B*, it is assumed that there is a local resource management instance in the RAN (i.e. an UMTS RNC), which knows the load of all cells and the signal quality for all users. For each user, this controller calculates a score based on the signal quality of the radio link between the user and each cell and the load of the cell. If the score of a different cell is better by a certain delta than the user's current cell's score, the user will conduct a handover to the new cell (see Algorithm 3). There are no handovers between different RANs as long as no global resource management application (N-RM) is running. When users leave the coverage area of the current RAN they will lose their sessions and scan for a new *RAN* as soon as they notice that the old one is no longer reachable.

When a user decides to create a session, it has to pass an admission control. The admission control checks, if there is enough capacity in the cell to allow the session, otherwise the session is denied.

In case that an N-RM is active, it monitors load and signal quality of the user's devices. Assume that a user approaches the border of his current cell. There is no other cell of the same RAN in range, so the local resource management has no possibility to improve the situation. Depending on the measurement strategy, N-RM might now be informed and decide to take action. N-RM does not know the exact location of the user and signal qualities between the user and other cells, it only knows which cells overlap with the user's current cell. Further, it aims for sending the user into a cell with low load. Therefore, it has to request load information from the overlapping cells if there is no sufficiently current data in the local cache.

With this information, N-RM will sort the cells by load and send the sorted list of cells to the user's device. The UE will then try the alternative cells in the given order and make a handover if the signal quality is sufficient and if admission control for the user's possibly active session succeedes.

The other situation which is handled by N-RM is an overloaded cell. Again N-RM might be informed about this in time depending on the chosen measurement strategy. If N-RM decides to take action, it fetches the list of users of the cell and the load of all other cells that overlap with it. Then it creates a list of recommended cells, ordered by load, and sends this list to a subset of the users in the cell (i.e., to 10% of the users). Again the users will change to a recommended cell if possible.

13.3 Results of N-RM Experiments

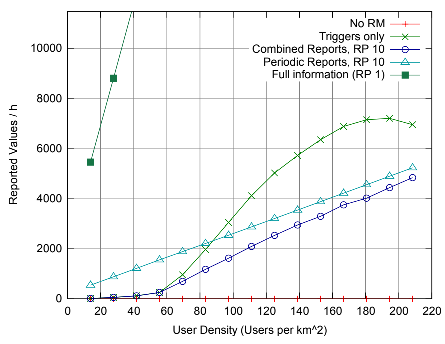


Figure 13.3: Number of transmitted values.

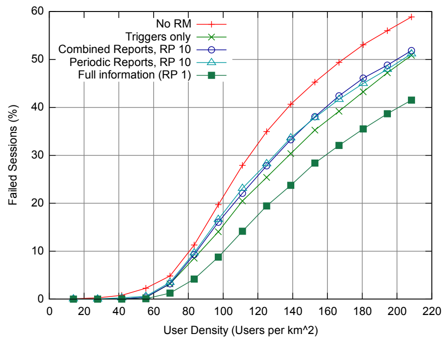


Figure 13.4: Failed sessions.

In this section the results of our simulations are described. The values have been normalized to 1 hour of simulated time and 1 km^2 wherever possible.

Figure 13.3 shows the amount of traffic which is produced by GMI reports. On the horizontal axis we have increased the user density and, therefore, the offered load.

One can see that the curve for periodic reports with a report period of 10 GP is linear in the number of users as expected. The curve for a report period of 1 GP is also linear but comparably high.

When the system load is low, triggers produce only very few messages as almost no critical events happen. However, when the load increases, the number of trigger-messages explodes. As one can see, the number of messages decreases again when the load is extremely high, as the parameters always stay above their thresholds.

The "combined reports" curve shows the result of an N-RM algorithm which switches to periodic reports after receiving a trigger for a subject. As one can see this curve always produces less messages than the triggered or periodic curves.

The quality of the decisions is evaluated using Figure 13.4, which shows the percentage of failed sessions. Here one can see that the three "realistic" systems are in the middle between the resource management with perfect information and the complete lack of resource management.

One astonishing fact is that the trigger curve is better than periodic reports and combined reports. This was not expected as the used trigger algorithm is quite primitive, it only acts when it gets an ascending trigger and does not control the success of its actions. However, the variance in the measured parameters is high, so the resource management is still triggered very often — which causes N-RM to send more recommendations to the users.

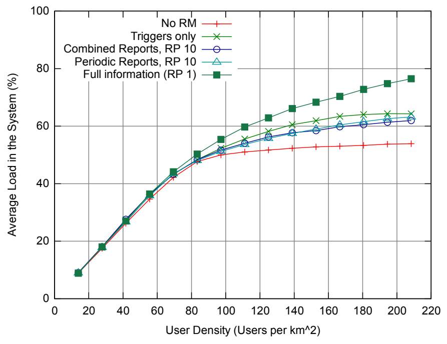


Figure 13.5: Average load.

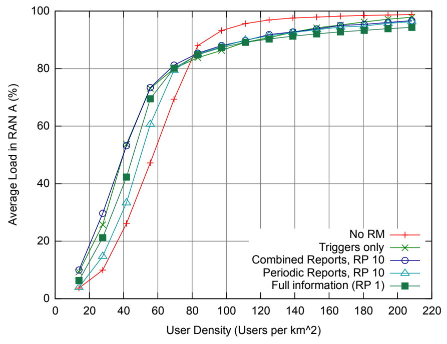


Figure 13.6: Load in *RAN A*.

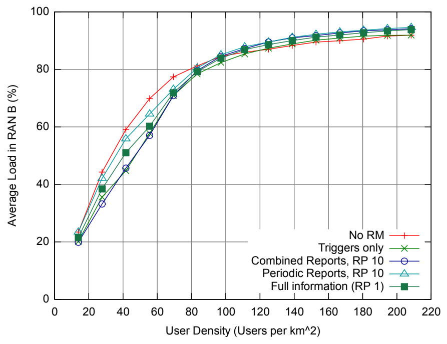


Figure 13.7: Load in *RAN B*.

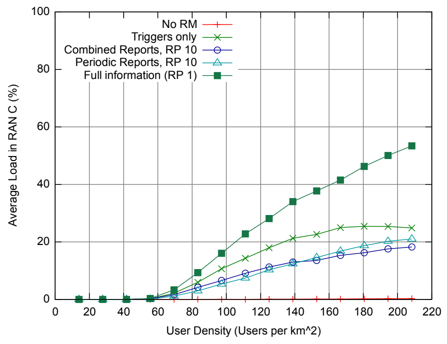


Figure 13.8: Load in *RAN C*.

Figure 13.5 shows the average load in the system. Again the user density can be seen as a measure of the offered load, while the vertical axis is the actual load which could be handled by the RANs. All five curves are very similar as long as the number of users is low. At around 80 users per km^2, the curves split. Without a resource management, the system is almost saturated at this point. At 210 users per km^2, the system runs at 54% load. The actual realistic systems are able to run the system at 62% to 64% load, again the triggered resource management achieves the best results. Using the resource management with perfect information, the system runs at 76% load with 210 users per km^2.

RAN A (Figure 13.6) which consists of one large cell with high capacity is always well-utilized for geographical reasons. The curve for "No RM" crosses the other curves. When the overall load is low, N-RM moves users from *RAN B* into *RAN A* when they experience bad signal quality in the center of the map. With a high user density, the load in *RAN A* becomes very high, so N-RM moves users away from *RAN A* — mostly to *RAN B*. As expected, *RAN B* which has a higher overall capacity shows the opposite effect, but on a smaller scale (Figure 13.7).

The most interesting curve is the load in *RAN C* (Figure 13.8) which is equivalent to WLAN. Without resource management, the users will never switch to one of the very small *RAN C* cells but stay in *RAN A* or *RAN B* even when passing by a *RAN C* cell. However, with N-RM, some users can be moved to *RAN C*, therefore, the available bandwidth of the whole network can be used better. This can be seen in Figure 13.8.

Finally, Figure 13.9 shows the number of handovers that occur in the system. Here one can see that a perfect resource management results in a very large number of handovers. This is mostly due to the event-triggered nature of the N-RM, which

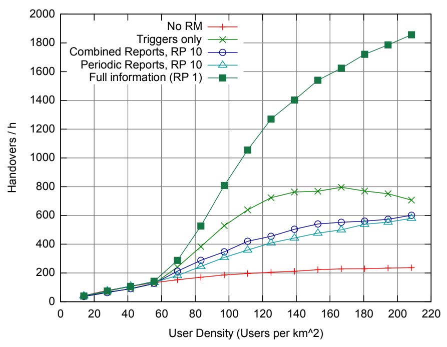


Figure 13.9: Number of Handovers.

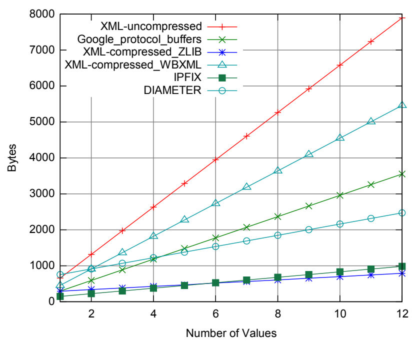


Figure 13.10: Data volume when transporting measurement data using different protocols.

takes action more often if it gets more input data. This is another type of overhead which can be caused by N-RM besides the signalling data on the backhaul links. Again there is a trade-off between the benefits like better load distribution and these extra handovers.

13.4 Data Volume

Resource Management	Users per km^2	Failed Sessions	Data Volume per Cell
None	56	2.28%	0.00 kBit/s
Combined Reports (RP 10)	56	0.40%	0.01 kBit/s
Maximum Information (RP 1)	56	0.09%	0.64 kBit/s
None	112	28.01%	0.00 kBit/s
Combined Reports (RP 10)	112	22.09%	0.09 kBit/s
Maximum Information (RP 1)	112	14.16%	1.18 kBit/s

Table 13.3: Data volume vs. success

So far, we have only been talking about the number of values that were sent through the GMI. In this section we will discuss the volume of management data which has to be sent over the backhaul links for heterogeneous resource management.

In our implementation, the GMI uses an uncompressed XML data format. However, in a production environment it would be favourable to use more bandwidth-efficient protocols. Basically any protocol which transmits key-value-pairs is suitable for transporting the GMI messages.

Figure 13.10 shows the performance of six different candidate protocols when transmitting a representatively chosen GMI dataset. One should note that with each of the candidates there are almost unlimited possibilities to encode the data differently which of course affects the volume. It was tried to select a meaningful encoding and to be fair among the candidate protocols. It should also be noted that only application layer data volume is considered, there are no IP-headers and also no headers of the layer 4 protocols that might be used in combination with our six candidates (TCP, UDP, SCTP).

- Uncompressed XML has advantages in terms of transparency as the data is transmitted in plain text. However it wastes bandwidth on the expensive backhaul links.

- WBXML [74] is a standard by the Open Mobile Alliance, which replaces XML tags by shorter binary strings, but leaves the actual content of the tags unchanged. In absence of a DTD, it builds a string table from the tag names and uses references to this table afterwards, which was the case here. In our example the data has been compressed to 69% of the size of the original XML.
- Google Protocol Buffers [80] is another representation which preserves the hierarchical structure of XML. In the example the data was compressed to 45% of the original size. The direct comparison with WBXML may be a bit unfair, as the Protocol Buffers encoder was able to use meta-information on the structure of the document. With a DTD, we would expect WBXML to perform roughly equivalent to Google Protocol Buffers.
- Diameter [81] is shown in this comparison as it is a common accounting protocol in 3GPP networks. It is the first protocol in the comparison, which has to map the document structure to flat key-value-pairs. However using meta-knowledge about GMI messages, the whole information can be reconstructed at the receiver. With our example data, the data volume used by Diameter was 25% of the original XML.
- IPFIX [82] is based on Cisco's Netflow v9 protocol. It is basically meant to export traffic flow data, but it is flexible enough for our purpose. IPFIX also works on flat attribute-value-pairs, but on the link it separates the attributes from the values. The attributes are sent once in so-called template records in the beginning of the transmission, while the values are sent separately in data records. It is also possible to use options templates for redundancy reduction, so values like long strings that appear in the data quite often only have to be sent once. Therefore IPFIX can save bandwidth compared to Diameter, our example-data was compressed to 12% of the original size.
- The last candidate protocol is a simple LZ77-zipped [83] version of the original XML data. The messages have not been compressed one by one, but the state on both sides is held during the transmission of multiple messages. This method of transmission is very efficient. In the long term, after 50 messages, the data volume could be reduced to 7% of the original XML size. However this advantage comes with increased costs in terms of memory and CPU-usage at sender and receiver.

For Table 13.3 the results from Section 13.3 have been combined with knowledge about message sizes. Here we assume data transport by IPFIX, which was the second-best solution in the comparison above, as we want to avoid the computational effort of compressing the data using LZ77. We also assume that each value is sent in a separate packet — which is a worst-case scenario — and add TCP and IP headers. As mentioned in Section 13.2.1, our granularity period is 15 seconds.

As we can see, heterogeneous access management gives a high benefit while using very little bandwidth. At 56 users per km^2 in the system the result with our combined periodic and triggered reporting is almost as good as the result obtained with the maximum available information.

At 112 users per km^2 the radio network is already overloaded as we can see from the success rate of the sessions. However, the number of messages we need for N-RM still remains negligible compared to the amount of user data transferred through

HSDPA or LTE cells. Here one could even consider sending all available data (GP 1) to the N-RM, which produces 14 times the data volume of the combined reports method, but still stays around 1 kBit/s per cell.

13.5 Relation to Mobile Botnets

In Chapter 10 possible denial-of-service attacks by mobile botnets were discussed. The target of these attacks were the cells of the mobile operator network.

With today's technology, a mobile network operator's options for responding to this kind of threat would be very limited. Today's networks support moving users between the cells of the same access technology for load reasons. The general effectiveness of this load-balancing strategy against this type of attack has been explored in Section 10.3.3.

The GMI would allow for new defenses during a denial-of-service attack by bots on the customer's devices:

- The resources of heterogeneous access networks network could be used more efficiently. A GMI-based Network Resource Management could make use of all available access technologies to distribute the load better. With the assumption that the worm is a userspace application on the mobile device which has no low-level control over handovers, this would already counter this denial-of-service attack very well.
- The GMI is not only suitable for mobility management, data collection for a mobile intrusion detection system would be possible as well. A large number of similar high-volume sessions that start at the same time would trigger an alarm, closer investigation would give the operator direct information about the reason for the traffic increase. So the operator would be able to apply countermeasures like temporarily disabling the user's account or rate-limiting their bearers.

A full analysis of attacker/defender interaction in this scenario would be an interesting future work.

13.6 Conclusions

Today, users of mobile networks increasingly demand data services and voice-calls for flat prices. This makes business difficult for network operators, there is hard competition on the market and revenues are shrinking. Operators have to cut costs — one way to do so is increasing network efficiency. This requires heterogeneous networks, as different access technologies have different strengths and weaknesses. It also leads to a need for new methods of data collection, as smart management is needed to take advantage of the different networks.

We have presented the Generic Metering Infrastructure, an information distribution system for mobile networks that has been developed with heterogeneous mobility management in mind, but which is general enough to support other applications. The GMI is based on a modified Publish/Subscribe System which employs subject-based addressing using domain names and subject-based filtering.

Optimized information distribution mechanisms are applied to reduce the number of messages that need to be transported. When using the GMI for heterogeneous access management, measurement data has to be sent over expensive backhaul links. Therefore we investigated how data can be compressed to save costs here.

We have shown that the flexibility provided by the GMI gives advantages when making heterogeneous handover decisions. It is possible to take good decisions with fewer data by switching between periodic reporting and setting triggers. Our simulation results show that customer experience can be enhanced significantly, while the cost in terms of produced overhead are comparably small.

The concept of the GMI is not only suitable for heterogeneous access management, but also for general management or security tasks (e.g. distributed intrusion detection). If the GMI was to be used for those purposes as well, the gains could be even bigger thanks to the “late duplication property” of the GMI’s publish/subscribe system.

Key findings in Chapter 13:

- **Mobile networks can greatly benefit from network-assisted handovers. The increased network performance comes at very low costs in terms of extra measurement data on the backhaul link.**
- **Smart monitoring using the GMI can reduce this extra measurement traffic. In a scenario with moderate load, the handover failure rate increased only from 0.09% with perfect information to 0.40% using the GMI configured for a combination of triggers and periodic reporting. At the same time, the amount of measurement data could be reduced by a factor of 64.**

13.7 Acknowledgements

The author would like to thank Vladimir Maldybaev for the results in Section 13.4 and Morten Schläger and Christian Hoene for fruitful discussions.

Part IV

Conclusion

14. Conclusions

Resource management problems are common in computer networks. For this thesis, two selected topics on resource management were investigated. Part I of this thesis laid the fundamentals for that and discussed related work.

In Part II of this thesis we looked into strategic resource management decisions that arise in the context of denial-of-service attacks. The results of this research are summarized in Section 14.1.

Part III was about resource management decisions in future mobile networks. It is advantageous to make heterogeneous handover decisions on the network side — opposed to letting the mobile device decide alone — to improve the utilization of the air interface. For this however, measurement data needs to be transported to the operator's core network which uses bandwidth on the backhaul links. These links are already a bottleneck, even without this extra traffic. The trade-off between handover efficiency and the volume of measurement data was investigated. The results are summarized in Section 14.2.

14.1 Denial-of-Service Attacks and Defenses

Before actually working on DoS defenses, we first looked into real-world denial-of-service attacks. In Chapter 2 we investigated the history of these attacks and common defense mechanisms. During this investigation we found the following facts:

Key findings in Chapter 2:

- **Denial-of-Service Attacks are very common on the Internet, they were considered the most serious operational threat in a survey among ISPs.**

- **In 2009, the bandwidth of the strongest documented denial-of-service attack was 49 Gbit/s. However most attacks use far less bandwidth, the current trend is a shift from flooding to semantic attacks. For example the Slowloris attack works with partial HTTP requests and is very effective.**

- **Motivations of the attackers are extortion of businesses or the desire to make a political statement. Another common motivation — probably with younger script kiddies — is loosing in online games or being dissatisfied with the policies of computer game companies.**

- **By combining sufficient overprovisioning with filtering, a very effective denial-of-service defense can be achieved. Such methods make many potentially attractive targets like big company websites, government websites and critical Internet infrastructure like the DNS root servers robust against DoS attacks.**

- **Most successful DoS attacks target small businesses, less-defended government websites or private persons.**

Overprovisioning as a defense strategy turns denial-of-service attacks into a resource allocation problem from the defender's perspective. In Chapter 3 we investigated a denial-of-service attack against Microsoft that exploited a specific weakness in the company's network infrastructure. The critical DNS servers that almost all of Microsoft's public services depended on were placed behind a single router. This single point of failure collapsed under the attack, making virtually all of Microsoft's services unavailable.

Key findings in Chapter 3:

- **Whether or not a network can be DoS-attacked not only depends on the amount of resources which are available for defending it. The network topology and service dependencies are key factors, as they can create weaknesses that an attacker can exploit to amplify his attack.**

In Chapter 4 we looked at Game Theory, as it promises a deeper understanding of situations that involve strategic interaction. We found the Colonel Blotto Game, in which two generals have to distribute their forces among a fixed number of battlefields. The general with the larger force will win a battlefield, while the general who wins most battlefields will win the war. This model is applicable to the resource distribution problems of attacker and defender in our scenario.

Key findings in Chapter 4:

- **A sequential version of the Colonel Blotto Game can be applied to the DoS problem. In this case the defender acts first, as he has to choose a static configuration for his network. The attacker can observe this configuration and attack accordingly.**

- **This gives the DoS attacker a huge advantage over the defender since he can exploit weaknesses in the defender's network. With about equal resources, the attacker will always win.**

With these preliminary investigations in mind, we started working on denial-of-service attack scenarios. First we needed a tool to simulate the attacks and the effects of the defender's resource distribution. For this, the network simulator Flones 2 was developed, which is presented in Chapter 6.

For Chapter 7 we initially made some theoretical observations on attacker- and defender strategies. After that we developed a method for finding DoS weaknesses in network and service topologies. The network is simulated using Flones 2, during the simulation a *Good Client's* success in getting answers to requests is monitored. Using standard optimization algorithms, the attacker improves his attack. As a result, we can see the network's most vulnerable spots. We also tested an outer optimization loop for the defender to fix these problems, however in this case the runtime of the nested optimizations explodes.

Key findings in Chapter 7:

- **We use the Good Client's packet loss as a metric for an attack's success. When increasing the attack traffic to a service, this metric follows a concave curve: When the service is already overloaded, further increasing the strength of the attack gives the attacker less and less benefit.**
- **An attacker with limited resources will concentrate his attack to a single target. The stronger the attacker is, the more targets he wants to attack.**
- **If multiple services have to be defended and all of them are equally valuable, the defender should split the defensive resources equally.**
- **In the case that m services run on n physical servers with $m>n$, the attacker will first choose the servers to attack, depending on his resources. Then he will attack all services on the chosen servers.**
- **Weaknesses in networks can be found by optimizing the attacker's strategy. However, adding an outer optimization for improving the network leads to very long simulation times.**

Chapter 8 builds on the theoretic observations from Chapter 4, where we saw that the attacker has a huge advantage in the Blotto/DoS game, as he can react to the defender's resource distribution decision.

Therefore, for Chapter 8 we used virtualization to strengthen the defender, allowing him to move his services between his servers during the attack. We have shown that periodic re-allocation of services based on the current load situation is beneficial because of the attacker's limited information about the attacked network. After each move of a service the attacker has to search for a new attack strategy using random probing.

There is a large space of solutions for both, the attacker and the defender. Only a subset of these was investigated in this thesis.

Key findings in Chapter 8:

- **Moving services at runtime brings a great benefit against an attacker who is not aware of this and will therefore not react effectively.**
- **A smart attacker can greatly reduce the benefit of this method.**
- **As a defender, it is dangerous to base decisions on the attacker's past strategy. The performance of the new strategy against its own best response should be the main criterion.**
- **Testbed experiments have shown that VM migration during a DoS attack is practically feasible and can be expected to bring a benefit as long as attackers do not adapt.**

Often DoS victims start thinking about defense only after the attack has already started. In this case a proxy-based solution like the one introduced in Chapter 9 is helpful. Our method is purely based on HTTP redirects as a proof of work, it does not require JavaScript or Captchas. We have shown that the method is effective if the defender has enough proxies and can therefore claim the resource advantage. However when purely looking at the effectiveness of the proxies, a JavaScript-based computational puzzle is superior.

Key findings in Chapter 9:

- **The designed proxy system is effective if the defender has enough resources. However if the use of JavaScript is possible, computational puzzles should be preferred.**
- **Load-balancing acts on information from the past. Therefore it has to be used carefully, a smart attacker can exploit its properties.**
- **The DoS model used in Section 9.5 does not include timing effects. Real-life experiments support the simulation results, but depending on the situation, considering increased delays due to the DoS attack might be important.**

Finally in Chapter 10 we have shown that botnets consisting of mobile devices can DoS the cells of a cellular mobile network. In the considered attack, the infected mobile phones are coordinated to strike only when their numbers are high enough to actually cause damage.

We have shown that this attack is feasible, but it is not clear if there is sufficient incentive for an attacker to implement this attack. However network operators should be prepared for the worst-case.

Key findings in Chapter 10:

- **The investigated DoS attack on the individual cells of mobile networks is technically feasible and a threat to mobile network operators. With central coordination by a botmaster, the mobile devices will only strike when they can actually cause harm.**
- **The purely decentralized variant of this attack, which is based on ad-hoc networks, has been found to be ineffective. However, this is not a 100% reliable result, as humans tend to cluster more than the randomly distributed users in the simulation. During events which are visited by thousands of people, critical masses of users might still be reached.**
- **It is not clear, what the incentives of an attacker who builds such a worm would be. However, the same can be said about the Stuxnet worm ([59]).**
- **One possible countermeasure is load balancing by moving users to neighboring cells. This improves the situation as long as those cells still have free capacities.**
- **The best defense is the prevention of malware-spreading on mobile phones. This will be harder as these devices become more and more open.**

As load balancing has shown to be an effective countermeasure, this investigation is one possible motivation for employing advanced resource management strategies in mobile networks. Such methods are investigated in *Part III* of this thesis, which is summarized in the following section.

14.2 Data Collection for Heterogeneous Handover Decisions

Handovers between different access technologies, i.e. GSM/UMTS and WLAN, are usually triggered manually by the user today. For increasing the network's efficiency it would be desirable to make handover decisions in the operator's core network. This would make it possible to assign each user to an optimal cell in terms of radio conditions and cell load.

The drawback of a centralized approach is that metering data about the situation in the radio access networks has to be transported to the handover decision engine that resides in the operator's core network. This measurement data has to be transported over the backhaul links between the base stations and the actual packet core. Unfortunately, the measurement traffic has to compete with user data on these links, and the backhaul is considered to be a major bottleneck in mobile networks for years to come.

In Chapter 12 we have described the Generic Metering Infrastructure, a modified publish/subscribe system for future mobile networks. The GMI is able to efficiently provide information about the network state, using triggers, periodic measurements or in a classic request/reply fashion.

In Chapter 13 we have set up a simulation of a mobile network consisting of three radio access technologies. This simulation has shown that network-assisted handovers as suggested by Fan et. al. in [63] are feasible when the required measurement data is transmitted efficiently by the GMI and that the benefits outweight the additional costs induced by measurement traffic on the backhaul-link.

Key findings in Chapter 13:

- **Mobile networks can greatly benefit from network-assisted handovers. The increased network performance comes at very low costs in terms of extra measurement data on the backhaul link.**
- **Smart monitoring using the GMI can reduce this extra measurement traffic. In a scenario with moderate load, the handover failure rate increased only from 0.09% with perfect information to 0.40% using the GMI configured for a combination of triggers and periodic reporting. At the same time, the amount of measurement data could be reduced by a factor of 64.**

Appendix

A. Flones 1

In this chapter the description of the original Flones network simulator is provided. As already mentioned in Chapter 6, Flones had a number of interesting concepts. However, it was more an "implemented formal model" than a network simulator. Therefore, it was impractical to use and had to be replaced by Flones 2.

A.1 Reasoning vs. Simulation

There are were two basic approaches to consider, for developing a tool that investigates networks for DoS vulnerabilities. The tool could do "reasoning" on the input parameters or perform some kind of network simulation.

A "reasoning" tool works as follows: After receiving the input, the tool constructs an appropriate internal model of the situation. This model is a static snapshot of the network state; it does not contain a concept of time. In this static model, different attack vectors are explored to find the best possible one. Approaches like this have already been used to find configuration problems in networks of Unix PCs, which allow users to increase their access permissions [84]. The difference to a simulation tool is that the later one has a concept of time. Events can trigger other events. This concept is more similar to classic network simulation. At first glance it seems like the static tool is much simpler. However in our context, flows can depend on each other and with a static tool it is not clear how circular dependencies can be resolved.

Figure A.1 shows an example of a circular dependency. Three data streams of 80 bandwidth units each pass through a triangle of routers each of which can handle 100 bandwidth units. It is not clear from the start, what the result of this situation should be. Assuming that all data streams lose an equal percentage of the incoming bandwidth in an overloaded router, the situation converges towards the blue numbers. In the general case it is not clear that there is any convergence at all. A static tool which explores networks for DoS-vulnerabilities would require a mechanism for

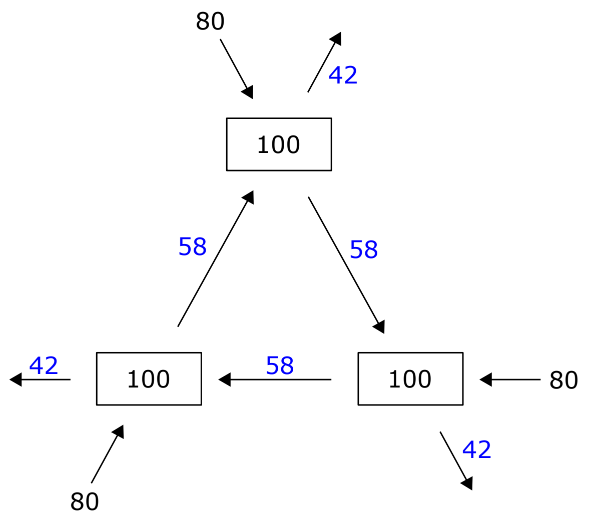


Figure A.1: A circular dependency.

handling such situations. From a network simulator one would expect that it behaves similar to reality in this situation — as time dynamics is considered, we do not need convergence to an equilibrium. One can say that a simulation tool which considers time dynamics appears to be closer to reality and is probably more intuitive to use, as it behaves similar to common network simulators and can generate a real-time output of the current simulation state. Therefore a simulation-approach was chosen for Flones.

A.2 Basic Concepts

Flones 1 has two basic ideas:

- Time is divided into rounds. All events that occur within one round are assumed to happen simultaneously.
- Services consist of state machines that communicate with each other. We aggregate multiple state machines to process aggregates of packets.

In Flones 1 terminology, a *flow* is an aggregate of packets with the same source node and the same destination node (where source and destination may be aggregated nodes, i.e. 1000 clients may be represented by a single simulation node) that have been sent in the same round. An alternative terminus would be packet aggregate or message aggregate — the later one was chosen for the Flones 2 terminology. However, as this section describes Flones 1, the original terminology is used.

A flow may consist of packets that belong to the same connection, i.e. 1000 RTP packets that belong to a video streaming session, however, it may also consist of 1000 TCP SYN packets that obviously belong to different sessions.

On both endpoints of a connection, finite state machines are used to keep track of the connection status. The state machines are aggregated the same way as packets, so each state in the state machine is annotated by a number that indicates the number of connections in this state.

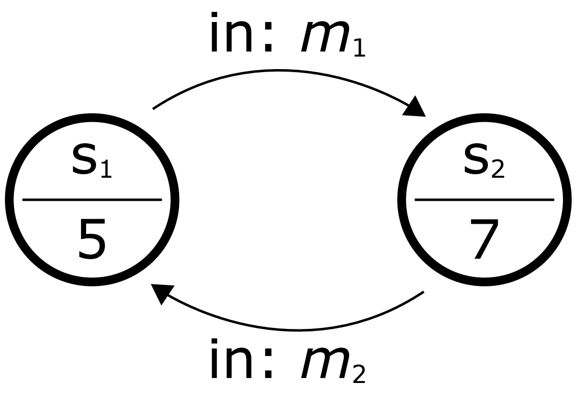


Figure A.2: A finite state machine with two states.

Figure A.2 shows such an *Aggregate State Machine* (ASM). There are two states s_1 and s_2, with five connections being currently in state s_1 and seven connections in state s_2. An incoming packet of type m_1 causes one *transition*, one connection switches from s_1 to s_2. Transitions also trigger outgoing packets. Usually packets arrive in form of a flow aggregate which contains multiple packets of the same type, i.e. a flow aggregate consisting of four m_1-packets would cause four connections to transition to state s_2 and possibly a new flow of multiple packets would be triggered.

A.3 Resources

There is a finite number of resources $R_1 \ldots R_n$ (i.e. CPU, network bandwidth, RAM). Connections that are in a specific state can use resources (i.e. a TCP connection in state ESTABLISHED required a certain amount of RAM). Further, transitions also use resources. Each node has a fixed amount of resources to spend per round.

If a resource is exhausted in one node, the simulator has to decide about the consequences. An algorithm that selects flows to be dropped must be independent of the request-order. All requests within a round occur logically simultaneously. The actual order in which they arrive is determined by simulation-internal mechanisms that should not affect the result.

If a flow aggregate A of 1000 flows arrives at node N and a flow aggregate B of 10 flows arrives "later" in the round, the 1000 earlier flows should not completely block the 10 flows of aggregate B. The logic behind this is that each simulation round stands for a real time interval. If a simulation round is 1 second long, flow aggregate A may consist of flows arriving with a rate of 1 flow per millisecond, while flow aggregate B consists of flows arriving each 100 ms. So the flow aggregates have not arrived one after the other but their individual flow arrival times were distributed within the time interval that forms the round.

In Algorithm 6 all states/flow aggregates that require a resource equally loose connections/flows if this resource is overloaded. This calculation is performed before committing the transitions in the finite state machine. The loss is calculated in two steps, first the resource usage by the states is calculated. Only if resources are available after this, transitions are considered. The idea behind this is that a node can be overloaded in a way that it cannot accept any incoming flows.

Algorithm 6: The constraint checker calculates the consequences of overload at a certain node. For simplicity this considers only a single service at the node.

```
input  : states, potential transitions, available_resources
output : updated states, committed transitions
// First handle load by states.
foreach $r \in resources$ do
    $resource\_usage\_by\_states_r = 0$
    foreach $s \in states$ do
        $resource\_usage\_by\_states_r$ += $calculate\_resource\_usage(r, s)$
    if $resource\_usage\_by\_states_r > available\_resources_r$ then
        $factor = resource\_usage\_by\_states_r\ /\ available\_resources_r$
        foreach $s \in states$ do
            $state.remove\_flows(factor)$
// Then try to commit transitions using the remaining resources.
foreach $r \in resources$ do
    $resource\_usage\_by\_transitions_r = 0$
    $remaining\_resources_r = available\_resources_r$ - $resource\_usage\_by\_states_r$
    foreach $t \in transitions$ do
        $resource\_usage\_by\_transitions_r$ += $calculate\_resource\_usage(r, t)$
    if $resource\_usage\_by\_transitions_r > remaining\_resources_r$ - then
        $factor = resource\_usage\_by\_transitions_r\ /\ remaining\_resources_r$
        foreach $t \in transitions$ do
            $t.remove\_flows(factor)$
            $t.commit()$
```

A.4 Knowing the Previous state

In Flones 1, connections between nodes are not individually identifiable, but part of aggregates and handled by Aggregate State Machines. This leads to problems if multiple transitions expect the same message, as shown in Figure A.3. As both s_1 and s_2 expect a message of type m, it is not clear whether on arrival of such a flow aggregate, connections should be moved from s_1 to s_2 or the other way round.

Practically this means that when a TCP ACK arrives, Flones 1 is unable to distinguish between an ACK from TCP states SYN_RCVD to ESTABLISHED, from ESTABLISHED to ESTABLISHED, from FIN_WAIT_1 to FIN_WAIT_2 or from CLOSING to TIME_WAIT.

This was not intended, but is a side-effect of the model. Accuracy was traded for a simulation speedup and this information got lost in the process. The idea behind these flows is still that they are individual sessions which do something individually, i.e. some TCP sessions between a web browser and a web server.

There are several possible solutions to this problem:

- One could give up determinism and choose the session for each incoming message m by chance among the connections that are currently able to process a message of type m.

  There is one strong argument against this. The method would mean that each individual flow is randomly assigned a session upon arrival. This can either

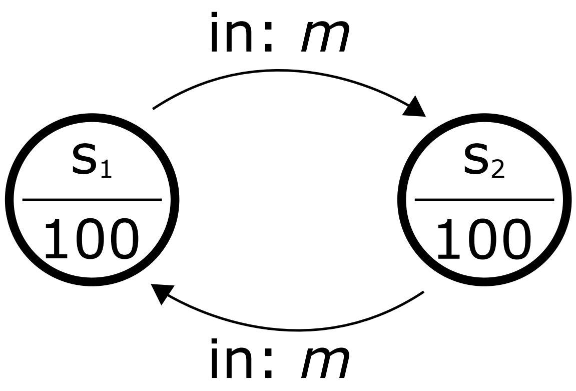


Figure A.3: Two transitions expect the same message.

happen uniformly (all available connections have the same chance of being assigned to the new flow) or according to any other probability distribution. Assuming that we have a protocol state machine as shown in Figure A.3, with 100 sessions in state s_1 and another 100 sessions in state s_2. Also assume that the transition from s_2 to s_1 is computationally expensive, while the transition from s_1 to s_2 is not. Now 100 flows with message m arrive. One could now assume that 50 flows change from s_1 to s_2 and 50 flows vice versa. However, the goal of Flones is not the simulation of "random" or "normal" behavior, but attacks performed by a smart attacker. This attacker would know that he can cause much more load by sending flows that cause a transition from s_2 to s_1 than the other way round, so he would probably want to send 100 flows that cause a transition from s_2 to s_1. This is not covered by the random model.

- Include the information about the previous state explicitly in the flow. In the example above, an incoming flow aggregate would contain the information "Expected state at destination: $s_2 \to s_1$". This method has a number of drawbacks. It requires flows to contain state information, which contradicts the idea of a finite state machine. It also contradicts our every-day experience with many network protocols. Therefore it is unintuitive to use.

- Introduce a way to uniquely identify flow aggregates. Not individual flows, but flow aggregates would be associated with an identifier. The states would keep a mapping which assigns a session counter to each flow aggregate identifier. This way, modeling the network with flow aggregates could be done in a rather intuitive way. A session identifier can be thought of as an "IP-5-Tuple", even though it does not identify a single connection but multiple connections.

  In the above example, each message of type m would be annotated by an ID, e.g. $m^{(A)}$. If the flow with ID A is currently in state s_1, incoming messages $m^{(A)}$ would cause it to transition to state s_2.

  This also has obvious disadvantages. It reduces the amount of aggregation, and it implicitly assumes that the same fate applies to all connections with the same ID. What should happen if $m^{(A)}$ in the example only contains 50 flow aggregates. To avoid ambiguity, the remaining flows would have to be dropped, there is no way of handling them e.g. in the next round.

When writing Flones 1 simulations, usually the second option was chosen. However, the combination of Aggregate State Machines and having to think of the state at the destination when sending a message turned out to be unintuitive and complicated. This is the main reason why Flones 1 was rarely used.

After recognizing this drawback, the conceptual work on Flones 2 was started to create a much more usable network simulator which is described in Section 6.

Bibliography

[1] Arbor Networks, "Worldwide infrastructure security report," 2009.

[2] J. D. Howard, *An analysis of security incidents on the Internet 1989-1995*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1998.

[3] R. Bush, D. Karrenberg, M. Kosters, and R. Plzak, "Root Name Server Operational Requirements." RFC 2870 (Best Current Practice), June 2000.

[4] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring internet denial-of-service activity," in *ACM Trans. Comput. Syst.*, vol. 24, pp. 115–139, New York, NY, USA: ACM, 2006.

[5] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," in *SIGCOMM Comput. Commun. Rev.*, vol. 34, pp. 39–53, New York, NY, USA: ACM, 2004.

[6] P. Vixie, G. Sneeringer, and M. Schleifer, "Events of 21-oct-2002," tech. rep., ISC, UMD, Cogent, 2002.

[7] J. Xu and W. Lee, "Sustaining availability of web services under distributed denial of service attacks," in *IEEE Trans. Comput.*, vol. 52, pp. 195–208, Washington, DC, USA: IEEE Computer Society, 2003.

[8] X. Liu, X. Yang, and Y. Lu, "To filter or to authorize: network-layer dos defense against multimillion-node botnets," in *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, (New York, NY, USA), pp. 195–206, ACM, 2008.

[9] J. Mirkovic, *D-WARD: Source-End Defense against Distributed Denial-of-Service Attacks*. PhD thesis, University of California, Los Angeles, 2003. Chair-Gerla, Mario and Chair-Reiher, Peter.

[10] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for ip traceback," in *In Proceedings of the 2000 ACM SIGCOMM Conference*, pp. 295–306, 2000.

[11] J. Ioannidis and S. M. Bellovin, "Implementing pushback: Router-based defense against ddos attacks," in *In Proceedings of Network and Distributed System Security Symposium*, 2002.

[12] T. M. Gil and M. Poletto, "Multops: a data-structure for bandwidth attack detection," in *SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium*, (Berkeley, CA, USA), p. 3, USENIX Association, 2001.

[13] A. Hussain, J. S. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in *SIGCOMM*, pp. 99–110, 2003.

[14] A. D. Keromytis, V. Misra, and D. Rubenstein, "Sos: Secure overlay services," in *In Proceedings of ACM SIGCOMM*, pp. 61–72, 2002.

[15] A. Stavrou, D. L. Cook, W. G. Morein, A. D. Keromytis, V. Misra, and D. Rubenstein, "Websos: An overlay-based system for protecting web servers from denial of service attacks," in *Elsevier Journal of Computer Networks, special issue on Web and Network Security*, vol. 48, 2005.

[16] R. Thomas, B. Mark, T. Johnson, and J. Croall, "Netbouncer: Client-legitimacy-based high-performance ddos filtering," in *Proceedings of DISCEX III*, pp. 14–25, 2003.

[17] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker, "DDoS Defense by Offense," in *ACM SIGCOMM 2006*, (Pisa, Italy), September 2006.

[18] A. Mahimkar and V. Shmatikov, "Game-based analysis of denial-of-service prevention protocols," in *CSFW '05: Proceedings of the 18th IEEE workshop on Computer Security Foundations*, (Washington, DC, USA), pp. 287–301, IEEE Computer Society, 2005.

[19] E. Kaiser and W. chang Feng, "mod_kapow: Mitigating dos with transparent proof-of-work," in *CoNEXT 2007 Student Workshop*, CoNEXT.

[20] T. A. Timothy, T. Roscoe, and D. Wetherall, "Preventing internet denial-of-service with capabilities," in *SIGCOMM COMPUT. COMMUN. REV*, p. 6, 2003.

[21] K. Argyraki and D. R. Cheriton, "Network capabilities: The good, the bad and the ugly," in *ACM Hot Topics in Networks (HotNets) Workshop*, 2005.

[22] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *CoNEXT '09: Proceedings of the 5th international conference on Emerging networking experiments and technologies*, (New York, NY, USA), pp. 1–12, ACM, 2009.

[23] W. Eddy, "TCP SYN Flooding Attacks and Common Mitigations." RFC 4987 (Informational), Aug. 2007.

[24] C. Meadows, "A cost-based framework for analysis of denial of service in networks," in *J. Comput. Secur.*, vol. 9, pp. 143–164, Amsterdam, The Netherlands, The Netherlands: IOS Press, 2001.

[25] J. V. E. Mölsä, "A taxonomy of criteria for evaluating defence mechanisms against flooding dos attacks," in *In Proceedings of the 1st European Conference on Computer Network Defence*, 2005.

[26] "HP OpenView." `http://openview.hp.com/`.

[27] P. Barham, R. Black, M. Goldszmidt, R. Isaacs, J. MacCormick, R. Mortier, and A. Simma, "Constellation: Automated discovery of service and host dependencies in networked systems," tech. rep., Microsoft Research, Cambridge, CAM, UK, 2008.

[28] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl, "Automating Network Application Dependency Discovery: Experiences, Limitations, and New Solutions," in *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation, 2008*, (San Diego, CA, USA), pp. 117–130, USENIX Association, 2008.

[29] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang, "Towards highly reliable enterprise network services via inference of multi-level dependencies," in *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, (New York, NY, USA), pp. 13–24, ACM Press, 2007.

[30] L. Popa, B.-G. Chun, I. Stoica, J. Chandrashekar, and N. Taft, "Macroscope: end-point approach to networked application dependency discovery," in *CoNEXT '09: Proceedings of the 5th international conference on Emerging networking experiments and technologies*, (New York, NY, USA), pp. 229–240, ACM, 2009.

[31] C. Probst, "Signalbasierte Analyse von Abhängigkeiten bei Netzwerkdiensten," Master's thesis, University of Tübingen, 2008.

[32] J. Schlamp, "Configuration management via frequency analysis of synthetic load fluctuations," Master's thesis, Technische Universität München, 2009.

[33] R. Gibbons, *A Primer in Game Theory.* Pearson Higher Education, June 1992.

[34] H. Gintis, *Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Interaction (Second Edition).* Princeton University Press, 2 ed., February 2009.

[35] J. Nash, "Non-cooperative games," in *The Annals of Mathematics*, vol. 54 of *Second Series*, pp. 286–295, Annals of Mathematics, September 1951.

[36] E. Borel, "La théorie du jeu et les équations integrales à noyau symmetrique gauche," in *C. R. Acad. Sci. Paris*, vol. 173, pp. 1304–1308, 1921.

[37] R. Golman and S. Page, "General blotto: games of allocative strategic mismatch," in *Public Choice*, vol. 138, pp. 279–299, 2006.

[38] B. Roberson, "The colonel blotto game," in *Economic Theory*, vol. 29, p. 24, September 2006.

[39] R. Powell, "Sequential, nonzero-sum "blotto": Allocating defensive resources prior to attack," in *Games and Economic Behavior*, vol. 67, pp. 611–615, 2009.

[40] V. Bier, S. Oliveros, and L. Samuelson, "Choosing what to protect: Strategic defensive allocation against an unknown attacker," levine's bibliography, UCLA Department of Economics, 2006.

[41] V. Misra, W.-B. Gong, and D. Towsley, "Fluid-based analysis of a network of aqm routers supporting tcp flows with an application to red," in *SIGCOMM Comput. Commun. Rev.*, vol. 30, pp. 151–160, New York, NY, USA: ACM, 2000.

[42] Y. Guo, W. Gong, and D. F. Towsley, "Time-stepped hybrid simulation (tshs) for large scale networks," in *INFOCOM*, pp. 441–450, 2000.

[43] B. M. Waxman, “Routing of multipoint connections,” in *Selected Areas in Communications, IEEE Journal on*, vol. 6, pp. 1617–1622, August 2002.

[44] Y. Gu, Y. Liu, and D. Towsley, “On integrating fluid models with packet simulation,” in *In Proceedings of IEEE INFOCOM*, 2004.

[45] A. Gitter, “Untersuchung von netzen auf dos-schwachstellen mit hilfe der flones simulationsumgebung.” Studienarbeit, August 2008.

[46] R. Ando, Z.-H. Zhang, Y. Kadobayashi, and Y. Shinoda, “A dynamic protection system of web server in virtual cluster using live migration,” in *DASC ’09: Proceedings of the 2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, (Washington, DC, USA), pp. 95–102, IEEE Computer Society, 2009.

[47] J. L. Wolf and P. S. Yu, “On balancing the load in a clustered web farm,” in *ACM Trans. Internet Technol.*, vol. 1, pp. 231–261, New York, NY, USA: ACM, 2001.

[48] S. Nakrani and C. Tovey, “From honeybees to internet servers: biomimicry for distributed management of internet hosting centers,” in *Bioinspiration & Biomimetics*, vol. 2, p. S182, 2007.

[49] A. Korsten, “Virtualisierung zur optimierung der verfügbarkeit bei dos-angriffen.” Studienarbeit, March 2008.

[50] Herley, Cormac, “So long, and no thanks for the externalities: the rational rejection of security advice by users,” in *NSPW ’09: Proceedings of the 2009 workshop on New security paradigms workshop*, pp. 133–144, New York, NY, USA: ACM, 2009.

[51] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1.” RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785.

[52] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1.” RFC 2068 (Proposed Standard), Jan. 1997. Obsoleted by RFC 2616.

[53] T. Vogt, “Simulating and optimising worm propagation algorithms,” 2003.

[54] Y. Bulygin, “Epidemics of mobile worms,” in *Performance, Computing, and Communications Conference, 2002. 21st IEEE International*, vol. 0, pp. 475–478, Los Alamitos, CA, USA: IEEE Computer Society, 2007.

[55] G. Yan, H. D. Flores, L. Cuellar, N. Hengartner, S. Eidenbenz, and V. Vu, “Bluetooth worm propagation: Mobility pattern matters!,” in *ASIACCS ’07: Proceedings of the 2nd ACM symposium on Information, computer and communications security*, (New York, NY, USA), pp. 32–44, ACM, 2007.

[56] J. Su, K. K. Chan, A. G. Miklas, K. Po, A. Akhavan, S. Saroiu, E. D. Lara, and A. Goel, “A preliminary investigation of worm infections in a bluetooth environment,” in *Proceedings of the ACM Workshop on Rapid Malcode (WORM)*, (Alexandria, VA, USA), 2006.

[57] B. Zhao, C. Chi, W. Gao, S. Thu, and G. Cao, “A chain reaction dos attack on 3G networks: Analysis and defenses,” in *IEEE INFOCOM 2009*, 2009.

[58] W. L. Tan, F. Lam, and W. C. Lau, “An empirical study on the capacity and performance of 3G networks,” in *IEEE Transactions on Mobile Computing*, vol. 7, pp. 737–750, Piscataway, NJ, USA: IEEE Educational Activities Department, 2008.

[59] N. Falliere, L. O. Murchu, and E. Chien, “W32.Stuxnet Dossier,” tech. rep., Symantic Security Response, October 2010.

[60] V. Gazis, N. Alonistioti, and L. Merakos, “Toward a generic "always best connected" capability in integrated WLAN/UMTS cellular mobile networks (and beyond),” vol. 12, pp. 20–29, 2005.

[61] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, “Performance anomaly of 802.11b,” in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, vol. 2, pp. 836–843 vol.2, 30 March-3 April 2003.

[62] 3GPP, “TS 32.401 V7.0.0: Telecommunication management; Performance Management (PM); Concept and requirements (Release 7),” 2006.

[63] C. Fan, M. Schläger, A. Udugama, V. Pangboonyanon, A. C. Toker, and G. Coskun, “Managing Heterogeneous Access Networks - Coordinated policy based decision engines for mobility management,” in *LCN ’07: Proceedings of the 32nd IEEE Conference on Local Computer Networks*, (Washington, DC, USA), pp. 651–660, IEEE Computer Society, 2007.

[64] A. Monger, M. Fouquet, C. Hoene, G. Carle, and M. Schläger, “A metering infrastructure for heterogeneous mobile networks,” in *First International Conference on COMmunication Systems and NETworkS (COMSNETS)*, (Bangalore, India), Jan. 2009.

[65] A. Carzaniga, “Architectures for an Event Notification Service scalable to Wide-area Networks.” PhD Thesis, 1998.

[66] G. Mühl, “Large-Scale Content-Based Publish/Subscribe Systems.” PhD Thesis, 2002.

[67] A. Zeidler, “A Distributed Publish/Subscribe Notification Service for Pervasive Environments.” PhD Thesis, 2004.

[68] J. Case, R. Mundy, D. Partain, and B. Stewart, “RFC 3410 - Introduction and Applicability Statements for Internet Standard Management Framework,” 2002.

[69] T. Bandh, “Automated Real Time Performance Management for Mobile Networks,” Master’s thesis, University of Tübingen, 2006.

[70] A. Monger, M. Fouquet, M. Schmidt, G. Carle, and M. Schläger, “A publish/subscribe system for heterogeneous access management, international patent wo/2009/112081,” 2009.

[71] R. Chalmers and K. Almeroth, “Developing a Multicast Metric,” in *Proceedings of IEEE Global Internet (GLOBECOM’00)*, (San Francisco, California, USA), Nov. 2000.

[72] 3GPP, "TS 23.003 v7.4.0: Numbering, addressing and identification (Release 7)," 2007.

[73] 3GPP, "TS 32.405 V7.4.0: Performance Management (PM); Performance measurements Universal Terrestrial Radio Access Network (UTRAN) (Release 7)," 2007.

[74] B. Martin and B. Jano, "WAP Binary XML Content Format." http://www.w3.org/TR/wbxml/, 1999.

[75] H. J. Wang, R. H. Katz, and J. Giese, "Policy-Enabled Handoffs Across Heterogeneous Wireless Networks," in *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, (Washington, DC, USA), p. 51, IEEE Computer Society, 1999.

[76] E. Stevens-Navarro and V. Wong, "Comparison between vertical handoff decision algorithms for heterogeneous wireless networks," in *Vehicular Technology Conference, 2006. VTC 2006-Spring. IEEE 63rd*, vol. 2, pp. 947–951, May 2006.

[77] R. Pries, A. Mäder, and D. Staehle, "A Network Architecture for a Policy-Based Handover Across Heterogeneous Networks," in *OPNETWORK 2006*, (Washington D.C., USA), Aug 2006.

[78] Tölli and P. A.Hakalin, "Adaptive load balancing between multiple cell layers," in *Vehicular Technology Conference, 2002. Proceedings. VTC 2002-Fall. 2002 IEEE 56th*, vol. 3, pp. 1691–1695 vol.3, 2002.

[79] IEEE 802.21, "Media Independent Handover Services." http://www.ieee802.org/21/, 2007.

[80] Google Inc., "Protocol buffers." http://code.google.com/apis/protocolbuffers/.

[81] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko, "RFC 3588 - Diameter Base Protocol," 2003.

[82] B. Claise, "RFC 5101 - Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information."

[83] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," in *IEEE Transactions on Information Theory*, vol. 23, pp. 337–343, IEEE Computer Society, 1977.

[84] D. Zerkle and K. Levitt, "Netkuang: a multi-host configuration vulnerability checker," in *SSYM'96: Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography*, (Berkeley, CA, USA), pp. 20–20, USENIX Association, 1996.

List of Figures

List of Tables

ISBN 3-937201-21-1

ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)