Attack Resistant Services Delivery over the Internet

Antti Koskimäki

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 31.12.2018

Supervisor

Prof. Raimo Kantola

Advisor

MSc Hammad Kabir

Aalto University
School of Electrical
Engineering

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Antti Koskimäki

Title Attack Resistant Services Delivery over the Internet

Degree programme Computer, Communication and Information Sciences

Major Communications Engineering **Code of major** ELEC3029

Supervisor Prof. Raimo Kantola

Advisor MSc Hammad Kabir

Date 31.12.2018 **Number of pages** 135+11 **Language** English

Abstract

The expansion of the global Internet and the advent of Internet of Things (IoT) have made Denial of Service (DoS) attacks a real threat against web services, especially as these attacks are easily mounted and there may be monetary incentives to enact them. Therefore, there is a need for practical security solutions, built on existing Internet architecture, that would help against the DoS- and specifically against Distributed Denial of Service (DDoS) attacks that are even more dangerous, as they can harness large amounts of network resources against a single victim.

The Realm Gateway security software concept aims to help in this regard, and it is basically a combination of Domain Name System (DNS) server and Network Address Translation (NAT) service. This system protects the web service behind it from unwanted traffic by detaching the service to a private network address space and allocating access for the service with the DNS for clients behaving in a good manner while active client reputation monitoring is utilized.

The tests done in this thesis show that the Realm Gateway system works well against DoS- and DDoS-attacks in some cases and it offers good ideas in the network security and service availability context, but to make the system fully feasible in practice, the Realm Gateway software itself should be refined, updated and tested further.

Keywords The Realm Gateway, Network security, NAT, DNS, DoS, DDoS, BGP, TCP, IPv4

Aalto University
School of Electrical Engineering

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Antti Koskimäki

Työn nimi Hyökkäyksen Kestävien Verkkopalveluiden Tarjonta Internetin Kautta

Koulutusohjelma Tieto-, tietoliikenne- ja informaatiotekniikka

Pääaine Tietoliikennetekniikka **Pääaineen koodi** ELEC3029

Työn valvoja Prof. Raimo Kantola

Työn ohjaaja DI Hammad Kabir

Päivämäärä 31.12.2018 **Sivumäärä** 135+11 **Kieli** Englanti

Tiivistelmä

Maailmanlaajuisen Internetin laajentuminen ja asioiden Internetin esiintulo ovat tehneet palvelunestohyökkäyksistä todellisen uhan nykyisiä verkkopalveluita vastaan, varsinkin kun palvelunestohyökkäyksiä on helppo toteuttaa ja niiden tekemiseen voi olla rahallisia kannustimia. Siksi on tarvetta käytännöllisille turvallisuusratkaisuille verkossa, joita on rakennettu olemassa olevan Internetin arkkitehtuurin pohjalle. Näitten ratkaisuiden tulisi suojata erityisesti jakautuneita palvelunestohyökkäyksiä vastaan jotka ovat vaarallisia, koska ne voivat hyväksikäyttää isoa määrää tietoverkon resursseja yksittäistä uhria vastaan.

Realm Gateway-turvallisuusohjelmisto on suunniteltu auttamaan juuri edellä mainittujen uhkien torjumisessa. Perusteiltaan tämä ohjelmisto yhdistelee Network Address Translation-järjestelmien (NAT) ja Internetin nimipalveluja hoitavan Domain Name System-järjestelmän (DNS) toiminnallisuutta. Realm Gateway suojaa sen taakse sijoitettuja verkkopalveluita ongelmalliselta tietoliikenteeltä eristämällä kyseisen palvelun yksityisverkkoon ja jakamalla pääsyä palveluun vain asiallisesti käyttäytyville asiakkaille suojaukseen liitetyn DNS-palvelun ja NAT-osoitteenvaihdon avulla. Asiakkaiden käyttäytymisen seuranta perustuu heistä kerättyyn mainetietoon ja tätä hyväksikäyttäen esimerkiksi liikaa palvelua kuormittavat asiakkaat voidaan sulkea pois.

Tässä työssä tehdyt testit osoittavat että Realm Gateway-järjestelmä toimii hyvin tiettyjä palvelunestohyökkäyksiä vastaan ja tarjoaa hyviä ideoita verkkoturvallisuuden ja verkkopalveluiden saatavuuden viitekehyksessä. Tulisi kuitenkin huomioida että mikäli tätä järjestelmää halutaan hyödyntää kunnolla käytännön tilanteissa, itse Realm Gateway-ohjelmistoa tulisi kehittää ja testata pidemmälle.

Avainsanat Realm Gateway-ohjelmisto, Tietoturva, NAT, DNS, Palvelunestohyökkäykset, Jakaantuneet palvelunestohyökkäykset, BGP, TCP, IPv4

Preface

This master's thesis was written for the WIVE project, connected to the Aalto 5G research project in Aalto University, with the work starting on July of 2018 and going on to December of 2018.

First, I'd like to thank my thesis supervisor, Professor Raimo Kantola, for giving me the opportunity to do this master's thesis for the Aalto 5G project and thus prove the skills that I have learned on my long journey through the university life. I'd also like to thank him for his excellent commentary and help during the writing of this thesis and especially during the finalization process of the thesis document.

Additional big thank you goes to my thesis advisor, Hammad Kabir, as his help proved to be invaluable when writing this thesis. He had great patience in answering all my questions, ranging from complex to trivial, and working with him was a positive, pleasant experience overall. A thank you goes also to my WIVE project manage, Jose Costa-Requena, who offered good tips and comments in regard to writing my thesis.

Finally, I would like thank my parents for their unwavering trust in me and for the support they have given me throughout all my university studies.

Espoo, Finland, 31.12.2018

Antti J. Koskimäki

Contents

Abbreviations

3DES	Triple Data Encryption Standard
ACK	Acknowledge
adj-RIB-in	Adjacent Routing Information Base In
adj-RIB-out	Adjacent Routing Information Base Out
AES	Advanced Encryption Standard
API	Application Programming Interface
ARP	Address Resolution Protocol
AS	Autonomous System
ASCII	American Standard Code for Information Interchange
ASN	Autonomous System Number
BGP	Border Gateway Protocol
CA	Certificate Authority
CERT	Certificate
CHAP	Challenge-Handshake Authentication Protocol
CIDR	Classless Inter-Domain Routing
CNAME	Canonical Name
CPU	Central Processing Unit
CRL	Certificate Revocation List
CRLF	Carriage Return Line Feed
Cusum	Cumulative sum
DDoS	Distributed Denial of Service
DDNS	Dynamic Domain Name System
DNS	Domain Name System
DNSSEC	Domain Name System Security
DoS	Denial of Service
DS	Delegation Signer
DSCP	Differentiated Services Code Point
DTLS	Datagram Transport Layer Security
eBGP	External Border Gateway Protocol
ECMP	Equal-Cost Multi-Path
ECN	Explicit Congestion Notification
ECS	Domain Name System Extensions Client Subnet
EDNS(0)	Domain Name System Extensions (version 0)
EDoS	Economic Denial of Sustainability
FIB	Forwarding Information Base
FIN	Finish
FQDN	Fully Qualified Domain Name
GNFS	General Number Field Sieve
GRE	Generic Routing Encapsulation
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
iBGP	Internal Border Gateway Protocol
ICANN	Internet Corporation for Assigned Names and Numbers

ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IGP	Internal Gateway Protocol
IHL	Internet Header Length
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IRC	Internet Relay Chat
ISP	Internet Service Provider
LDDoS	Low-rate Distributed Denial of Service
loc-RIB	Local Routing Information Base
LXC	Linux Containers
MAC	Medium Access Control
MD5	Message-Digest 5
MitM	Man-in-the-Middle
MX	Mail Exchange
NAT	Network Address Translation
NIC	Network Interface Card
NS	Name Server
NSEC	Next Secure Record
OS	Operating System
OSPF	Open Shortest Path First
QoS	Quality of Service
P2P	Peer-to-Peer
PAT	Port Address Translation
PDoS	Permanent Denial of Service
PDU	Protocol Data Unit
PKI	Public Key Infrastructure
RIP	Routing Information Protocol
RR	Resource Record
RRSIG	Resource Record Signature
RSA	Rivest–Shamir–Adleman
RST	Reset
SFQDN	Service Fully Qualified Domain Name
SHA-2	Secure Hash Algorithms version 2
SHA-3	Secure Hash Algorithms version 3
SIP	Session Initiation Protocol
SLAAC	Stateless Address Autoconfiguration
SSH	Secure Shell
SSL	Secure Sockets Layer
SYN	Synchronize
TCP	Transmission Control Protocol
TLD	Top Level Domain
TLS	Transport Layer Security

TTL	Time-to-Live
UDP	User Datagram Protocol
VLSM	Variable-Length Subnet Masking (VLSM)
VM	Virtual Machine
VPN	Virtual Private Network
VoIP	Voice Over Internet Protocol

Figures

Code snippets, equations and tables

Code snippets

Equations

Tables

1 Introduction

With the growth of the Internet and with the advent of Internet of Things (IoT), there are increasing number of networked devices all around us. The combination of utilizing these devices and gathering data about people's communication and behavior in social networks, among other places, has led to new network service concepts. In addition, various existing services expand, update and see competitors emerge. This can be thought as a digitalization process, where many aspects of everyday life are coming to be handled or supported by these services. Examples of this merging of physical and digital realms can be smart homes where web service can adjust central heating automatically based on temperature measurements, self-driving cars that exchange sensor and location information with each other and remote control and monitoring of machinery in distant locations in an industrial setting.

There are various important questions regarding these new services such as if some of them would be necessary or beneficial at all, especially if one wants to protect privacy and maintain direct control instead of delegating responsibilities to artificial intelligence? It is possible, however, to see important service concepts where digitalization and networking can offer improved efficiency and safety, energy savings and decreased monetary costs. For example, utilizing banking services on the web can save time and resources compared to an actual visit to a bank office and using self-driving cars could negate the human errors of the drivers. When these services become critical for enabling everyday life to go on smoothly, one must ensure service security and reliability, as disregarding these qualities can cause devastating effects. It is easy to imagine that problems with automated car controls can cause traffic accidents and malfunctions in controlling industrial equipment remotely could cause loss of revenue and even human injuries in extreme cases.

The security of networked services relates to the realm of *computer security* and *cybersecurity* which are mostly interchangeable concepts. This context has three major aspects: confidentiality, integrity and availability. *Confidentiality* generally means access control and ensuring privacy whereas *integrity* means that accessed data is trustworthy. Lastly, *availability* means that services should be accessible and working in the time of need, hopefully without a fault. Availability can be connected to the idea of reliability where work can be done, for example, to ensure that physical components of the system are not faulty and have back-ups. In this thesis, in the realm of cybersecurity, availability is thought more as a quality linked to reacting either proactively or actively to threats from outside in order to keep a particular service up and running. Simple example of this for some network service is having measures available to handle sudden unexpectedly high amount of incoming clients without large effect on the overall system performance. All three of these major aspects are important for critical web services and disregarding one of them may lead to issues with the other two.[1] Major factors related to problems with service availability specifically can be *Denial of Service (DoS) attacks*, where unwanted and bogus clients overload the service. This can be connected to the increasing amount

of IoT-devices, which is expected to be nearly 10 billion in year 2020, as can be seen in Figure 1 [2]. Unfortunately, malicious entities may harness these devices as participants to DoS-attack campaigns as they might have poor or non-existent security features.

Making important web services truly feasible requires taking related cybersecurity concerns seriously. This can mean designing systems and devices with security in mind from the bottom up, even if it would require more resources. In the case of IoT, this could mean that even the simpler devices could have some access control and the possibility to update their software in case of security flaws. With networking infrastructure, key point with enhancing service security is to leverage existing technologies to limit the costs to various Internet Service Providers (ISP), who are important middle-men in the Internet, as they usually control the routes between the client and distant web services. ISPs are rarely direct stakeholders with these services so network updates and improvements that affect the connections between the clients and servers are more likely to go through if they don't demand much from a particular ISP, especially in financial terms.

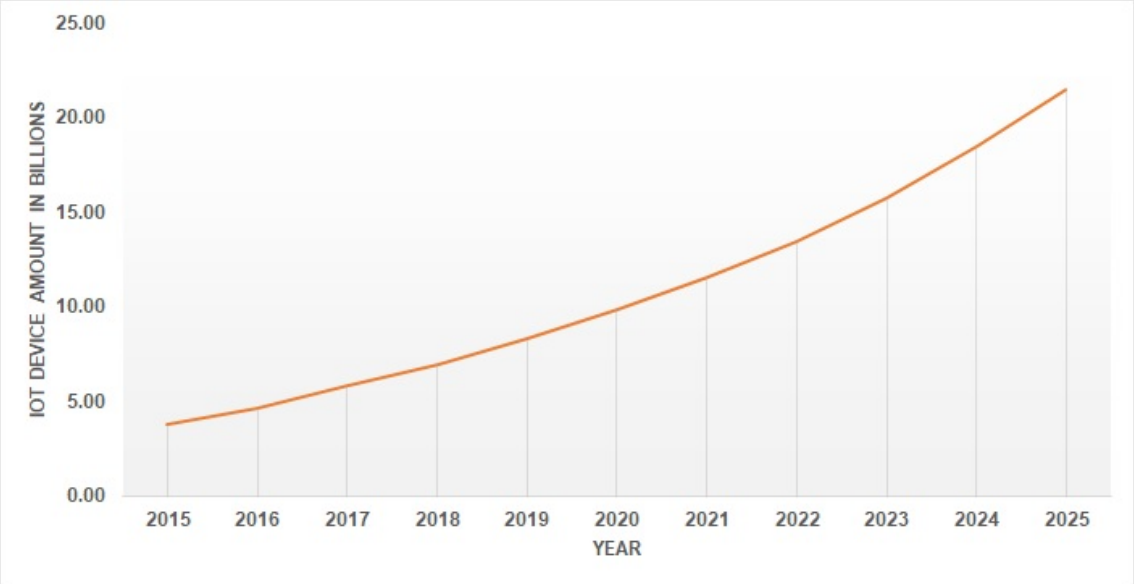

Figure 1: Projection on the global amount of connected IoT devices

1.1 Background and motivation

The basic routing of data in the Internet is done by the *Internet Protocol (IP)*, from which two versions are in use: Internet Protocol version 4 (IPv4) which uses 32-bit addresses and Internet Protocol version 6 (IPv6) which uses 128-bit addresses, both of which define the addressing scheme of the whole network. *IP addresses* are analogous to real life post addresses, where these addresses are added to packets, or digital

IP packets with networking, to denote their destination and the source they came from. Additionally, packets are directed towards the correct destination by entities en route, which would be postal logistic centers, harbors, airports, etc. in physical realm and router computers in the network in the IP case.[3][4][5]. As nowadays there are more and more networked devices in existence, one can notice that the IPv4 address space of roughly 4,2 billion addresses is not even enough for distributing personal addresses for each human on earth. This limitation becomes more apparent when non-personal connected devices are added to the calculation. IPv6 was devised in parts to solve this addressing limitation as it increases the address pool size to 2 to the power of 128, but it has not yet wholly replaced IPv4 as implementing the overall change could be expensive and difficult for ISPs. One way to circumvent the IPv4 addressing limitations is to utilize *Network Address Translation (NAT)* methods, where multiple network entities are placed behind a single IP address [6]. Even if IPv4 is still the mainstay on the Internet, NATs and IPv6 have made it easier to increase network size, which is apparent with the growing amount of IoT devices.

Keeping important web services up and running reliably becomes challenging if they face DoS-attacks or *Distributed Denial of Service (DDoS) attacks* where bogus clients may be distributed all over the Internet, which makes them harder to pinpoint; to clarify, DDoS-attacks are DoS attacks with multiple attacking network hosts, whereas DoS-attack would usually refer to a case with singular attacking node or to the overall concept of malicious resource denial. Malicious entities can benefit from the increased amount of available IP addresses due to NATs and IPv6 and can gain access to large pools of clients, which can be used in the DDoS-campaigns. The service downtime caused by these attacks can lead to the host suffering large monetary damages due to lost revenue while also inconveniencing legitimate clients. As nowadays many enterprises rely more and more on web services, the cost of even a single, effective DoS-attack campaign could be millions of dollars [7]. One example of serious successful DDoS-attacks in recent times is the attack campaign against the ISP Dyn in The United States in 2016, which utilized the Mirai botnet that was essentially a network of compromised devices with a central, malicious controller. The attack eventually caused various large web services such as Netflix and Twitter to be down for several hours.[8] Another recent example attack was the DoS-campaign against government-related web services here in Finland, which caused the national user verification system to be down. This made people unable to access services such as the website for Finnish National Insurance Institution which relates to the government-funded health care system.[9] When reasons behind all these attacks are speculated beyond cyber-vandalism, it could be surmised that sometimes certain state actors are in play when dissenting web services connected to political opponents or differing viewpoints are disrupted. Additionally, there is often financial gain to be had when criminal organizations target companies for ransom with the threat of impending DDoS-attack.

As various types of DoS- and DDoS-attacks can have large negative impact on service availability, it would then be prudent to devise ways to mitigate these attacks especially in regard to critical web services. If one aims to improve how networks work

instead of trying to manipulate each different web service specifically in the network infrastructure context, it is difficult and cumbersome to enact large scale changes, especially if these changes are dependent on complex new hardware and software changes and updates. This is related to the fact that Internet is a series of different networks, where each network has a its own operator who have varying motives and financial resources. Having this large group of network actors then leads to a big problem of overloading web services in the global Internet as IP source address validity is not enforced. Similarly to postal services, the source address of an item, be it digital or physical, can be forged as it is usually not needed for the item to reach the correct destination. Additionally, there are no global network identities that could be used to pinpoint malicious actors, which means that often temporary and likely fabricated IP source addresses are the only identifier of suspect data traffic. Internet service providers could mitigate source address forgeries by enacting ingress filtering to make sure that outgoing data traffic from their end has legitimate source addresses, but it demands resources and it is not often in their best interest in the financial sense [10].

When one is looking for solutions and improvements to web service security, it is wise to keep the aforementioned ISP constraints in mind. It would be prudent to build on top of the existing Internet infrastructure which means utilizing the standard tools of Internet routing and addressing such as *Domain Name System (DNS)*, IPv4, IPv6, NATs and *Border Gateway Protocol (BGP)* [3][11][12][13]. In addition, perhaps one way to enact positive change could be to try to adjust the network operator mindset so that the responsibility of cleaning up malicious data traffic is shared with incentives coming from a common pool of resources. This could mean having some sort of trust information available so that web services could make informed decisions about clients and not automatically expect them to behave in a benevolent manner from the get-go. In game theory, it is beneficial to act poorly versus one's opponent to maximize gains if trust and relationships between the players are not established yet, which can mean in a networking context that servers should be at least moderately cautious with incoming clients.[14]

For achieving better results in securing web services, it is also good to combine different approaches which can lead to adding previously mentioned trust concept to work with existing network protocols and technologies. An example of this is the *Realm Gateway* software concept which is designed to act as gatekeeper and as a gateway to outside Internet for important web services and mobile users. It maintains reputation data via monitoring client behavior which relates to trust management. Additionally, it acts as a firewall, as a NAT and as a DNS server to hide and protect critical web services from unwanted attention and also to facilitate legitimate connections between well-behaving clients and these services.[15]

1.2 Research problem

The goal of this thesis is to find out if a combined and creative use of standard Internet routing, NAT and DNS with the help of Realm Gateway software could be a feasible and practical way to improve web service security and availability especially against DoS- and DDoS-attacks. Methods for achieving this can include actively monitoring client behavior in the system to block out malicious entities as quickly as possible and distributing incoming load to the service to multiple entry points.

1.3 Objectives and scope

In this thesis, the scope of research and testing is limited to a simulated network environment, where traditional network infrastructure with clients, servers and routers is set up virtually utilizing mainly private network spaces. The focus will be on validating system functionality with legitimate network data traffic and sending simulated DoS- and DDoS-traffic to the system while observing how well it performs in defending against this bogus traffic in terms of resource use, blocking attackers and admitting real clients. Additionally, objective is also to observe how much running the defensive measures in the simulation will inconvenience legitimate users with delays.

1.4 Structure of the thesis

Following this introductory chapter, the thesis will have three chapters related to background information. The first of these will present the basic building blocks of Internet architecture and network routing with some added insight to security issues related to these concepts. The second background chapter will discuss concepts related to network security in the Internet in more detail. The focus there is to present threats to service availability and also to showcase the Realm Gateway software principles. Finally, the last background information chapter will discuss the principles of simulated network environments that will be used in the actual simulation scenarios in the thesis.

After the background chapters, there is a chapter explaining the simulation setups and testing scenarios where the Realm Gateway performance is observed with the aforementioned research problem in mind. In this chapter, the results of each test are also presented and analyzed in conjunction of the respective test scenario. The final chapter of the thesis which comes after the discussion about testing will then have conclusions about the tasks that have been done and musings about future work.

2 Internet architecture and routing primer

In telecommunication, when two different entities connect to each other with the purpose of exchanging data, there is a need to establish a common language and rules so that the communication can succeed and the messages are understood. This is where the concept of *protocols* comes in. To enable data transmissions between vast arrays of different programs and devices, various set of rules have been designed so that the entities in the network have shared knowledge on how to initiate communication, how to send and receive data and how to interpret this data. These rules are protocols such as IPv4, *Transmission Control Protocol (TCP)* and Hypertext Transfer Protocol (HTTP) [16][17].

Additionally, two major principles are in play when one discusses network data traffic. The data transmissions can be either *connectionless* or *connection-oriented*, where with the former, data is sent towards destination without initial connection setup between the sender and the final receiver and without guarantees that the data actually reaches the destination. With the latter, connection establishment procedures are used to ensure that the participants can reach each other before the actual data traffic can begin, where there are more options available for controlling the data flow. It is important to note that connectionless networking requires the use of existing addressing infrastructure, where entities in the network generally have global, unique addresses, which are utilized in *routing* so that data traffic can be directed toward correct nodes at least initially. In contrast, with connection-oriented networks, communication between network nodes requires signaling and network management and the address scheme linked to data traffic is local, where the identifying logic for the data transmission flows for example doesn't have information about the wider network structure.[11, pp. 1–33]

As the Internet is actually a collection of networks with countless different nodes, getting into contact with a peer will usually mean that the data is sent through intermediate machines that need to know where to direct this data. This forwarding of traffic to the correct destination is made possible by a process called *routing*. The actual role of routers is further defined by the *end-to-end principle* that is prevalent in the Internet. End-to-end means that as the traffic should flow from the initial source to the final destination, the control mechanisms for manipulating this traffic in case for transmission errors, network delays, etc. are placed to the end points as much as possible. This can reduce the processing load for intermediate routers, but the downside is that data traffic management responsibility is then shifted to the traffic source and destination, which can be especially problematic for web servers serving large numbers of clients.[11, pp. 1–33][18]

This chapter describes the basic building blocks of the Internet with the *TCP/IP model* and also discusses the standard routing principles with also providing insight into the DNS, which makes Internet more user-friendly by providing address translation from hard-to-read numerical addresses to more understandable text names.

2.1 The protocol stack in the Internet

As there are multiple different technologies and protocols in use in the Internet in parallel, it is paramount to use architecture where there are common elements in place to ensure network and protocol inter-operability. With this in mind, the TCP/IP model is used here to describe the basic structure of the Internet in a layered manner, where each layer has a set of protocols and technologies for specific, common purpose and is generally meant to work independently, utilizing primitives and abstractions to use the services of other layers.[19]

2.1.1 IP and TCP/IP model

The TCP/IP model, which is shown in Figure 2, describes an architecture where network entities are placed into a protocol stack, with the IPv4 and IPv6 being the critical component on the Internet layer that is the neck of the hourglass of the whole model. On the highest level, there are various applications such as web browsers and e-mail readers, which, in turn, can utilize TCP and *User Datagram Protocol (UDP)* on the transport layer below for controlling the flow of data transmission. This TCP/UDP data traffic can then use IPv4 and IPv6 beneath it for addressing and routing [20].[19]

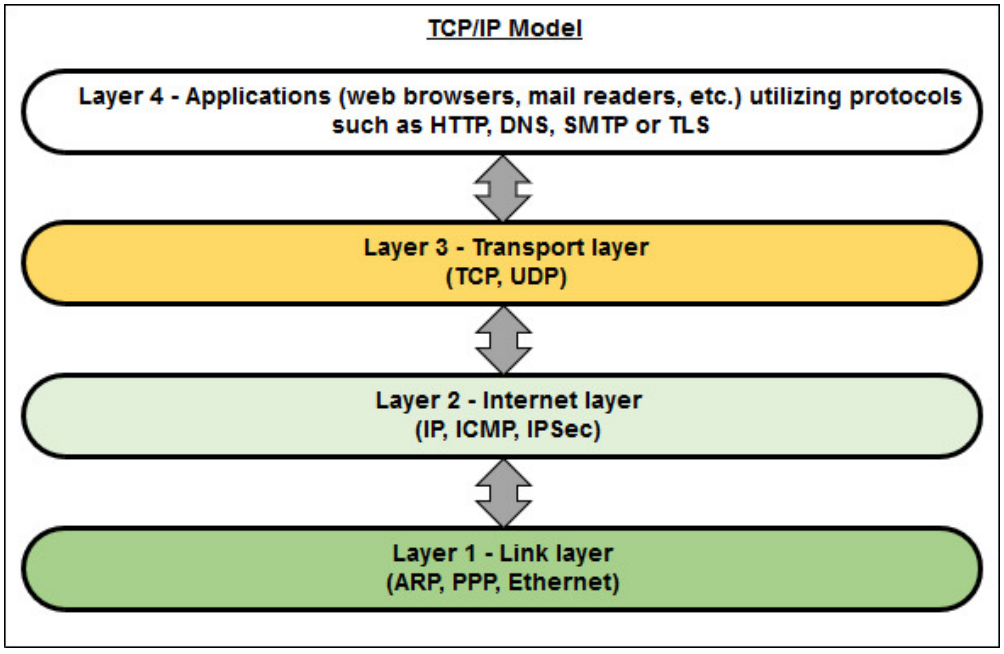

Figure 2: The TCP/IP network model

With TCP/IP protocols, transmitted items are usually data packets, such as IP packets or UDP packets, or IP and UDP datagrams for alternate term, and a common

acronym for these data transport units is Protocol Data Unit (PDU). On the link layer below IP, the computer handling the data transmission directs IP packets to an Ethernet connection or to wireless radio connection for example, as link layer has usually protocol access to the next entity or hop in the network. Link layer protocols, such as Institute of Electrical and Electronics Engineers (IEEE) 802.3 with Address Resolution Protocol (ARP) can direct data to this next hop via some physical medium such as data cable that is essentially abstracted to this layer [21].

The process of transferring data by the sender towards the destination is done by *encapsulation* of data where a protocol in a layer in the TCP/IP model adds header and possibly footer information to the data as it moves downwards. In reverse, when data is moved upwards from lower layers, the header and footer information is peeled off, layer by layer, by *de-encapsulation* until only application data is left. These headers and footers contain layer-specific control and addressing data that will actually enable the telecommunication. This process is illustrated by an example with the TCP/IP model in Figure 3, where some application data is sent through network utilizing UDP. The application PDUs are first prepended by UDP headers, then IP headers with addressing information and finally link layer frame headers and footers that contain information on how to reach the next hop in the network via physical medium. After possibly traversing through various network nodes that likely do de-encapsulation and encapsulation on their own, the receiver gets the sent data packet which will go through the de-encapsulation process that yields the relevant application data to the destination program.[19][3][11]

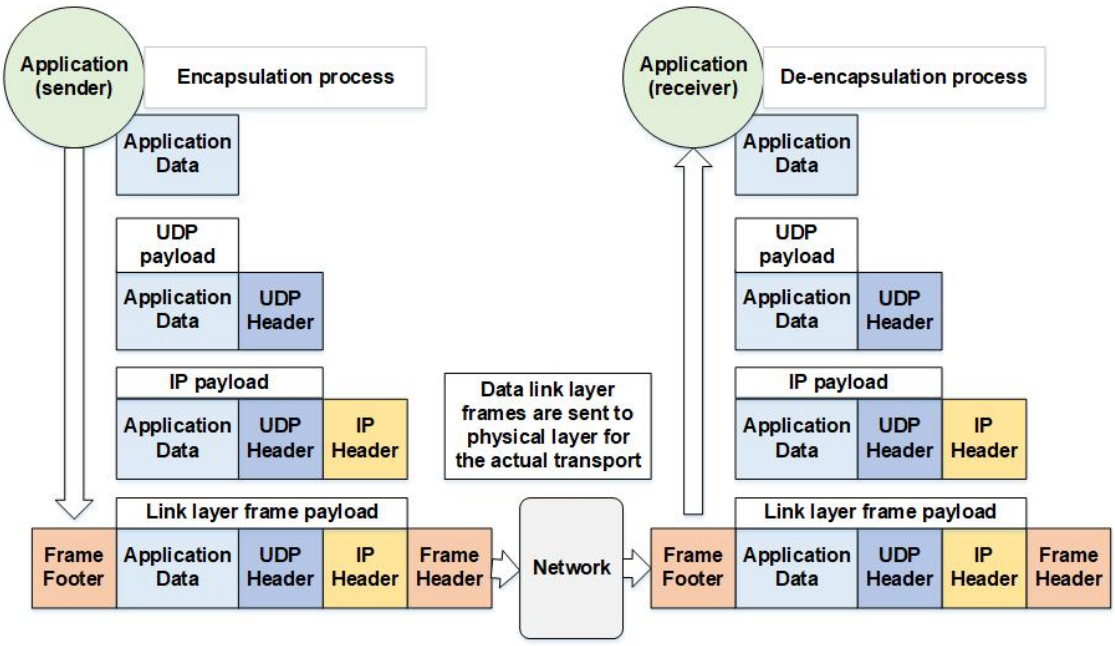

Figure 3: Encapsulation and de-encapsulation of network data

With the TCP/IP application layer abstraction, the application data can be arbi-

trary, and in practice it varies as web page content can be text, pictures or video, all of which can be transferred with HTTP for example on top of the lower layers. The application layer can also contain session and security management protocols such as Transport Layer Security (TLS) protocol which is used for data encryption and for access control. The application layer also includes sockets which act as a logical connection point between programs and the Operating System (OS) to enable network connections. [11][19][22]. In regard to more specific information about TCP, UDP and IP headers below the application layer, they can contain error-correction information and checksums to make it possible to notice data corruption or other problems that could happen on the route from source to destination. More importantly, IP headers and the frame headers on the link layer contain address information which will be used to look for the correct destination for a specific packet, either in the global Internet in the case of IP or in the local network in the case of frame headers which can utilize machine-specific Medium Access Control (MAC) addresses. [11, pp. 368–482]

There are some negative sides in practice related to relying just on IP on the Internet layer and doing de-encapsulation of data on the routers. As IP has spread all over the Internet, the limitations and problems it may have do affect pretty much everybody. This can be seen especially with IPv4 as it has only limited space for adding additional complexity and functionality to the protocol. It is also difficult and costly to do large upgrades to the Internet layer protocols as they must be enacted on so many entities. When data is forwarded deeper and deeper into the network towards distant destinations, there are no guarantees that all routers handling the data act optimally or even in a benevolent manner. It is possible that routers could de-encapsulate data further towards application layer to gather sensitive user information such as passwords as they certainly have the capability of parsing TCP, UDP and HTTP protocol messages for example, instead of just routing the data forward. To have some safeguards against this kind of behavior, it is generally not legal in Europe for the ISPs to read the payload of TCP and IP packets for example. With vast array of routers in the Internet, problems may emerge even if the routers are configured properly to do what they should. For example, routers may get congested and start dropping packets due to technical failures or abnormal amounts of data traffic.

It is also possible that security issues reside below the Internet layer, such as with ARP *spoofing*, where "to spoof" generally means to fabricate communication nodes' source addresses. A hostile entity could use spoofing to masquerade as another machine and receive data traffic that is intended to reach somebody else. The ARP spoofing specifically could be enabled by network nodes accepting and caching ARP address data too leniently, where attacker can inject falsified information to nearby network nodes.[23] These types of activities are usually called Man-in-the-Middle (MitM) that are also possible with IP, perhaps in conjunction with ARP spoofing. The malicious node could masquerade to be the IP destination in order to falsify data traffic and to draw out sensitive information from the sender. To defend against this, at least with the end-to-end principle, safeguards are generally expected to be at the

end points of a connection. Normal, connectionless IP data traffic requires that at least the originator of traffic uses some protocols or applications above the Internet layer if he wants to ensure that legitimate destination has been reached. The sending party can create, for example, a connection-based TCP session with the other end point where some assuring replies from the intended destination are received. On the other side, servers could also require these sessions to be set up so that they can ascertain client legitimacy with the ongoing request-response type messaging. As untrustworthy routing entities could wiretap ongoing data traffic in some cases, there are application layer protocols widely in use to boost telecommunication security. One example is the Hypertext Transfer Protocol Secure (HTTPS) which utilizes Secure Sockets Layer (SSL) security suite or its newer version TLS. HTTPS can be used to encrypt data traffic so it cannot be deciphered fully without proper decryption keys and also to authenticate the entity that is communicated with.[11, pp. 763–799][24]

2.1.2 IPv4 and ICMP

The basic addressing and routing on the Internet are based on IPv4 and IPv6 addresses, where a node in the network has one or more of these addresses so it is reachable. With the more commonly used IPv4, this address is a 32-bit value that is usually depicted as a set of four 8-bit values separated by dot notation. The four fields of an example IPv4 address ”123.71.0.226” have 8-bit values ranging from 0 to 255. Nowadays the IPv4 routing is done using Classless Inter-Domain Routing (CIDR) with Variable-Length Subnet Masking (VLSM) which introduced the CIDR-notation for depicting IP address ranges with subnet mask length. An example IPv4 address range ”123.71.0.0/16” in CIDR-notation would mean that subnet range contains 2 to the power of 16 or 65536 addresses as the number 16 behind the slash-character implies. In the simplest case, the mask size 32 denotes a single IP address. In routing decisions, different size net masks can be used in bitwise comparisons to check if the destination address belongs to a specific range of addresses and should therefore be directed towards the connection that is linked to this range of addresses.[4][11, pp. 572–601][25]

With IPv4, the address range is limited to 2 to the power of 32 (around 4,3 billion) different values, where several sub-ranges are reserved for special use. These IPv4 address ranges are presented in Table 1. To help circumvent this limitation, private networks can set up NAT boxes between their network space and the public Internet, where private addresses from the range ”10.0.0.0/8” for example, which are supposed to be kept out of the public Internet, are mapped to the public IP address of the NAT server. This is a very common usage case for private address ranges, but naturally there are other uses such as setting up totally detached private networks, where the private addresses are easily distinguishable from the public IP addresses. Additionally, reserved addresses include loopback addresses to help with system testing, multicast address range and ranges for specific test and demonstration setups, among other things.[26]

Table 1: Special IPv4 addresses

Address block	**Scope**	**Description**
0.0.0.0/8	Software	Current network descriptor
10.0.0.0/8	Private	Network address space for private use
100.64.0.0/10	Private	Shared communication address space between ISPs and clients
127.0.0.0/8	Host	Loopback address space
169.254.0.0/16	Subnet	Link-local addresses
172.16.0.0/12	Private	Network address space for private use
192.0.0.0/24	Private	IETF Protocol assignments
192.0.2.0/24	Documentation	Addresses for testing, documentation and examples
192.88.99.0/24	Internet	Reserved for future use
192.168.0.0/16	Private	Network address space for private use
198.18.0.0/15	Private	Reserved for testing
198.51.100.0/24	Documentation	Addresses for Testing, documentation and examples
203.0.113.0/24	Documentation	Addresses for Testing, documentation and examples
224.0.0.0/4	Internet	IP multicast
240.0.0.0/4	Internet	Reserved for future use
255.255.255.255/32	Subnet	Limited broadcast address

The PDU of IP is a data packet or datagram that contains an IP header at the beginning and the data payload after it, which is essentially the data to be transferred that is received from the upper TCP/IP model layers. The IPv4 header structure is presented in Figure 4. The most important field in the IPv4 header is the destination address, as routing is done using it. The additional relevant header fields are described below:

- Version – The version of IP the packet uses
- IHL – Internet Header Length (IHL) which denotes the length of the IP header
- DSCP – Differentiated Services Code Point (DSCP) which can be used for Quality of Service (QoS) control in some cases; if QoS-control is implemented, it can be used to differentiate packets into levels of resource access for example, where one type of packet is prioritized above others in the case of heavy traffic load
- ECN – Explicit Congestion Notification (ECN) which can be used to notify routers about network congestion

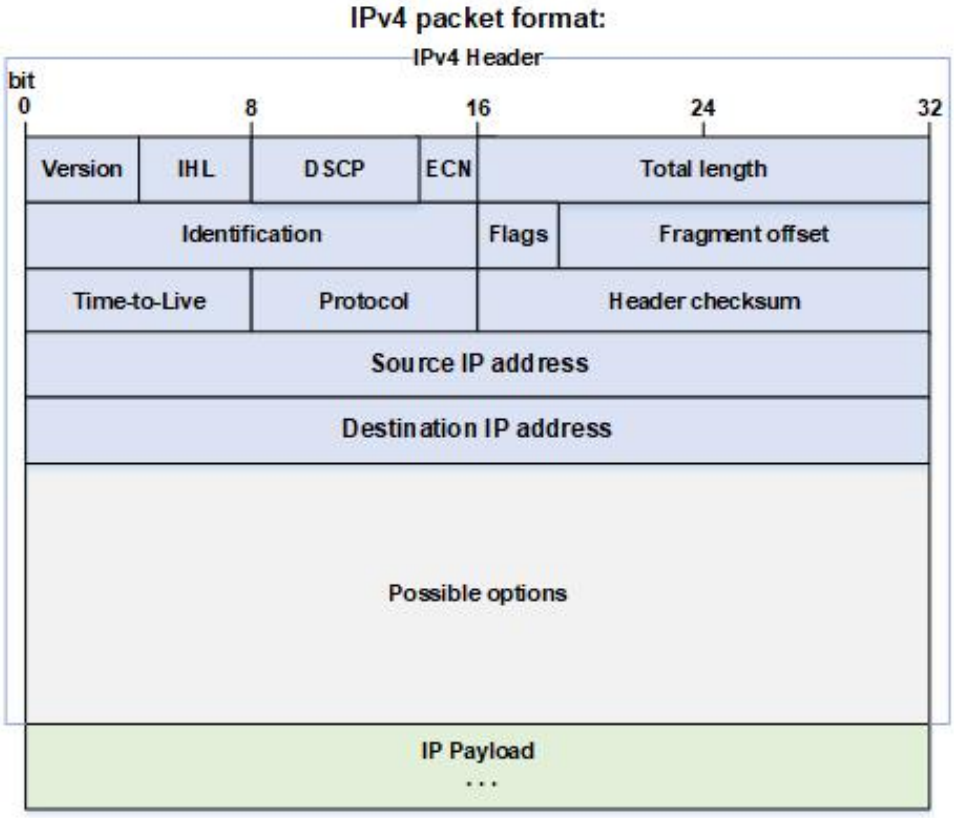

Figure 4: IPv4 packet and header format

- Total length – The length of the whole IP packet
- Identification, Flags, Fragment offset – These are used to identify and manage fragmented IP packets
- Time-to-Live – Time-to-Live (TTL) denotes how many hops the packet is allowed to travel before router drops it; this value is decreased by one by each router that handles the packet
- Protocol – Denotes the protocol of the payload such as TCP or UDP
- Header checksum – This is used to check that received header is valid
- Possible Options – These could be utilized for traffic engineering for example but they are rarely used.

The IPv4 header offers some options for managing traffic congestion and the QoS. The latter connects to network delays and bandwidth requirements and denotes a service level where a client for some specific service can use it with acceptable quality. The problem arises from the fact that utilizing these options may require support from the whole underlying system which may not be possible in diverse networks without central control. There is also a problem with the limited space in the IPv4

header, even with the possible additional options, which makes large scale extension of functionality difficult. It is good to notice that checking IPv4 header validity is done by calculating a checksum value from the received packet header and comparing this to the value in the header checksum field. This means that malicious router could manipulate the IP header beyond adjusting the TTL, calculate a new checksum to be added to the header and pass this packet onwards. Probably the largest issue with improper use of IPv4 is the possibility to change the IP source address to an arbitrary address, though, as the source is generally not utilized in the connectionless IP routing.[4][11, pp. 572–601]

As IP itself doesn't offer that much support for monitoring network status, the Internet protocol suite includes Internet Control Message Protocol (ICMP) that is used mainly as a diagnostic tool to help with network infrastructure upkeep where ICMP messages can notify network nodes about problems or malfunctions. Very common use for ICMP is that intermediate network nodes send ICMP replies to standard IP data packet traffic instead of routing it forward if they encounter errors. The originator of the IP traffic can then react to these replies when it can identify the problem from the ICMP message error codes. Even though ICMP is considered to be an Internet layer protocol, it works over IP, but it is usually handled more directly by the particular network node's OS. ICMP responses could be sent quickly based on preset procedures without forwarding data to sockets for example. ICMP can be very helpful when setting up routing infrastructure and private networks to ascertain connectivity as it can be used to send echo messages by ping-command with most OSes, where basic ICMP reply is expected, if there is an IP connection between the nodes.[27]

In regard to IPv6, there was and is a need to upgrade the basic Internet building block of IPv4 due to protocol design issues such as the limited global IPv4 address range. IPv6 was developed to solve these problems with having the option to run IPv4 and IPv6 in parallel during transition and upgrading period, where tunneling techniques can be used to send traffic through mixed networks. This process of replacing IPv4 with IPv6 altogether has been quite slow, however. Even at the time of writing this thesis, 20 years after the IPv6 inception, the IPv6 deployment rate is still only 25% which denotes the ratio of devices that advertise IPv6 connectivity in the global Internet [28]. The IPv6 adoption rate is increasing slowly, though. Basically, IPv6 offers far larger address space and extensible IP packet header structure that enables far more protocol options than IPv4. As is discussed later in this chapter, IPv6 does suffer from possible source IP address forgery similarly to IPv4, so this thesis uses IPv4 in the discussion and testing for simplicity. Possible solutions here can then be expanded to IPv6.

2.1.3 TCP, UDP and network ports

As IP is a connectionless protocol, ensuring reliable connection in some manner is left to the transport and application layers above the Internet layer. Applications and devices that utilize Internet vary, however, so there are lots of uses for less reliable and thus less demanding communication schemes too. Additionally, there is the need to differentiate multiple services that are available behind a single IP address. For this, and for establishing common practices if and where certain types of services are available on a web server, computer operating systems utilize port numbers that range from 0 to 65535. New connections to a server are made using pairs of port numbers and IP addresses, which direct the connection to the correct service within. With the TCP/IP encapsulation, the destination and source port numbers are included in the transport layer header data. The IP address related to the web service and the port are then bound to specific program sockets on the application layer, and these sockets can direct data traffic to the correct program. When describing IP addresses and related ports, the port number is often tagged after the IP address, separated by a colon-character.[11, pp. 34–96]

Web servers and related services which are for specific purpose are usually set to listen established port numbers which are coordinated by the Internet Assigned Numbers Authority (IANA), so that clients generally know which port to connect to when trying to reach a certain service such as HTTP server for example. The important transport layer protocols that carry port information in their headers are UDP and TCP, where UDP is connectionless and lightweight and TCP is connection-oriented, more reliable, but also more complex. Services that use well-established ports with UDP and/or TCP include Secure Shell (SSH) which mostly uses TCP and listens on port 22, DNS which usually uses UDP and listens to port 53 and HTTP which can use both UDP and TCP, depending on the demands of the content and connection, and listens to port 80 [29]. Setting the port to some predetermined value is often not required on the client side, as clients can choose an arbitrary, available port number for the connection in their respective system, especially when multiple, simultaneous client-server type connections are to be made to various outside servers. As an additional note, continuous packet flows from a single source towards some destination that utilize some common protocol such as UDP or TCP are often called traffic flows or network flows, from which a term flow is a shorthand [30].

Using UDP and TCP works in the same manner as using IP in the sense that they add header information to some upper level payload data, which is then used in the encapsulation and de-encapsulation process. The comparison of UDP and TCP headers is presented in Figure 5. The UDP header is quite simple as it only has source and destination port numbers, length denoting how long the whole data packet is and optional checksum field that may not even be in use with IPv4. This makes UPD packets fast to process, but dealing with possible errors such as lost or delayed UDP packets or corrupted data is left to upper layer protocols and applications. These entities may then skip doing any corrections at all, if certain percentage of

lost packets would not affect the quality of service noticeably. Perhaps the most common uses for UDP are simple one request-one answer type services such as DNS [12], where both requests and replies are quick to process and just re-sending queries will usually solve issues caused by lost packets.[20]

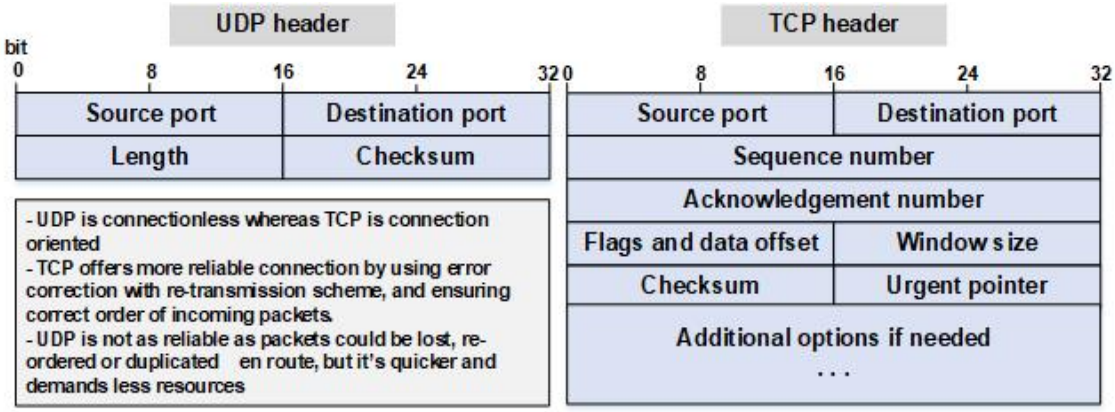

Figure 5: UDP and TCP headers

Often network services demand that the transmitted data is without faults, arrives in correct order and does so in a limited time-frame, all of which place more demands on the protocol in use. Generally, in the transport layer, TCP fills this role. As is presented in Figure 5, the TCP header has a structure which includes fields for data sequencing, error checking with checksum and overall connection management and flow control with various flag options and protocol timing window size adjustment. The basic idea of TCP is to be a connection-oriented, stream-based protocol, where connection setup is done with the initiator sending TCP messages with Synchronize (SYN) flag on, in the Flags and data offset field, to notify that it wants to establish connection. The receiver would then respond by sending a TCP message back with the SYN and Acknowledge (ACK) flags on to note that it is ready to proceed. Finally, the initiator should then reply with a TCP message with the ACK flag up, and after this is received, the actual data transmission can proceed. This procedure is presented in Figure 6. TCP uses the acknowledgement number and sequence number fields to ensure connection validity, as, for example, the protocol checks with these fields that the final ACK reply from client is connected to the correct SYN-ACK message. The TCP connection tear-down has similar request-response procedure with different flag values, whereas the actual data transfer happens in a streaming manner, where a sequence of data packets is sent, in order, towards the destination. The sequence numbering and checksums are used here to notice errors and if, for example, one data packet is corrupted, the receiver can ask for a re-sending of this data that has the specific sequence number. In addition, the window size field and acknowledgement messages during the stream are used to manage the rate of data transmission, as the connection may get congested or one endpoint may need to limit incoming traffic if it suddenly needs to process something else.[11, pp. 602–620][16]

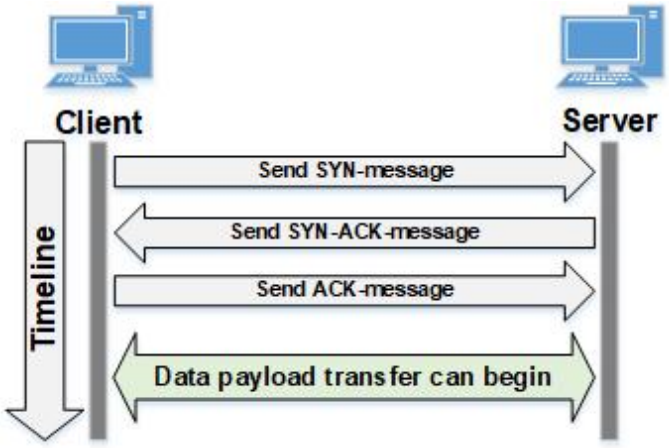

Figure 6: TCP connection establishment

In the security context, with connectionless UDP on top of IP, the issue of source address spoofing remains, as UDP isn't concerned about the IP attributes on the lower layer. This means that servers that offer UDP-based services may receive bogus traffic, where the originator of these messages doesn't care about the reply, as it doesn't usually even have access to the forged IP source address. This kind of traffic, which is essentially a part of a DoS or DDoS attack, can be problematic to defend against. Attackers can vary the IP source address which makes filtering just few source addresses inefficient if one wants to weed out malicious clients. One countermeasure would be in these cases to block incoming UDP traffic altogether, but that will block legitimate users as well. On the other direction, UDP itself doesn't offer any protection against entities on the data traffic route that sniff out packet contents and perhaps send forged replies to UDP messages if they are able to respond before or instead of the actual target UDP server.[31]

As TCP utilizes the three-way handshake when setting up the connection, there is an assurance that the initiator of the connection is actually there, if he responds to the SYN-ACK message with ACK. TCP too has the downside of receiving bogus traffic, though, as malicious clients could just send SYN-queries to try to drain server resources without handling or caring about the SYN-ACK responses. Another downside of TCP is the additional complexity compared to UDP, where establishing and maintaining TCP connections uses more computational resources, which can make DoS- and DDoS-attacks towards TCP servers even more damaging. Similarly to UDP, TCP traffic can be examined and parsed by malicious entities on the communication route, and on there, the possibility exists to disrupt or even hijack the TCP stream if an eavesdropper injects packets to the stream that match the sequence number in use at that point [32]. For resolving the security risks involved in unknown entities de-encapsulating and exploiting data payloads, both UDP and TCP generally rely on upper level protocols for data encryption and for authentication.[11, pp. 763–799]

2.2 Routing in the Internet

In the global Internet, there are often lots of intermediate nodes between peers that wish to communicate with each other. The Internet addressing is done with IP, and the IP addresses are then used in routing decisions to direct traffic to a correct destination. This task is the responsibility of routers which are usually intermediate servers on the network, managed by ISPs, and designed for this specific purpose. The working principle of routers is that they reside on the borders of different network segments while having interfaces connected to each segment. Generally, within a small local network or network segment, the data is forwarded to destination by broadcasting queries on the link layer about who has the destination address, for example with ARP in the case of small Ethernet networks. When affirmative reply is then received, the data can be sent to the destination directly. When a client within a segment wants to connect to a peer residing on a different network, the data is sent to a default router which will look up the destination address from the incoming IP data packet header and look for a corresponding entry in its *routing table*. This table contains information where to forward the packet based on its destination address, and the router can then forward the packet either directly to the destination, if it happens to be on the very next network segment, or to subsequent router which knows a route further towards this destination.[3, pp. 56–108]

As network sizes and demands on the routing procedures vary on the Internet, there are differing options available for the routing protocols. These protocols decide how to formulate and maintain the routing tables, how to direct traffic based on these tables and how to inform other routers about routing server status and routing table data changes if necessary. Generally, the routing process demands reliable exchange of network topology information between the routers, so request-reply type peer validation and error correction are in use for example with utilizing TCP for communication. Example protocols which are presented in this section to advance understanding about routing principles are Routing Information Protocol (RIP) and *Open Shortest Path First (OSPF) protocol*, which are generally for smaller scale networks, and *Border Gateway Protocol (BGP)*, which is used for routing between larger networks. Currently with RIP, the protocol version in use is 2, with OSPF it is 2 (for IPv4) and with BGP it is 4.[13][33][34]

2.2.1 RIP and OSPF routing

Although both RIP and OSPF are usually meant for small and medium scale networks, they do differ in working principle. Essentially RIP is simpler, less complex and uses distance vector routing, but it has several limitations. OSPF, on the other hand, is more complex and heavier link state protocol with added routing functionality. The main differences between the two are elaborated further in Table 2. One important common quality for most routing protocols is that they need to use IP-based communication to exchange network topology data which is the case for both RIP and OSPF and which also means that they are application layer entities in

some sense. [3, pp. 142–194][33][34]

Table 2: RIP and OSPF comparison

RIP - Distance vector	**OSPF - Link state**
Routing type: Hop count based	Routing type: Link cost based, where variables could be used to calculate routes beyond hop amount
Routing table: Contains hop counts to different destinations which are updated based on distance vector messages; these are periodically sent by neighboring routers and contain their current routing tables	Routing table: Routers have a link state database which has data about the whole network topology and the routing tables are calculated from this database; only changes to this database due to network changes are sent as updates to other routes
Max. network size: 15 hops	Max. network size: No strict limitations
Routing algorithm: Bellman-Ford	Routing algorithm: Dijkstra
Network topology: Flat	Network topology: Supports hierarchical networks
Resource use: Uses less memory and processor resources	Resource use: Uses more memory and processor resources
Convergence time: Slow	Convergence time: Fast

Showcasing RIP and OSPF illustrates differences on how routing information is set up and transferred between routers, but the exact workings of these protocols is not the focus of this thesis. RIP and OSPF are however discussed in more detail in Appendix A, mainly to offer contrast to BGP routing which is elaborated on later in this chapter.

It is also good to note routing in the cybersecurity context. No matter what routing protocols are in use, routers play a big part in functioning networks, as they are the central nodes through which most of the of data traffic flows through. Malfunctioning routers could mean for example that sections of the network are cut off or that parts of the network retain too much data traffic, where processing this traffic takes lots of resources. As both OSPF and RIP routing tables are updated by messaging over IP, there are potential security issues similarly to other communication processes over IP. The update messages could be eavesdropped, and it is possible that IP and ARP spoofing could be used to inject falsified messages to the network. If routing tables are compromised, the routing process as a whole suffers dramatically, as data could be routed to a *black hole* address which essentially leads to nowhere, or as all data could be routed specifically towards one destination to cause a DoS-attack. For security, OSPF and RIP for example can enable message authentication and encryption for routing table update messages which helps routers to reject false updates.

Routing software bugs and poor configuration could still lead to compromised routers, though, especially if the security settings are not managed properly. Additionally, as network bottlenecks, routers themselves are susceptible to DoS- and DDoS-traffic which could fill the routers' network links and would then delay, if not prevent, all ongoing legitimate data traffic.[35][36]

2.2.2 BGP

As OSPF and RIP have scale limitations, there is a need for a solution to connect multiple smaller network segments together to form the Internet structure of network of networks. The routing protocol for this purpose is BGP, which routes data between *Autonomous Systems (AS)* that are network areas containing sub-networks, where routing could be done with OSPF or RIP for example. Figure A4 in Appendix A illustrates how BGP router could be connected to OSPF routing infrastructure. There, the router acts as a gateway to outside Internet and directs traffic in by aggregating and advertising the internal network to its Internet routing peers. Especially with BGP routing and related address aggregation, this process involves sending out summarized entries about large network ranges that are called IP prefixes that describe the network part of the IP address utilizing the CIDR notation network masks.[3, pp. 239–279][13]

The BGP is based on path vector routing, which is an advanced version of distance vector routing that tries to negate loops while also limiting the routers' computational load when the network grows in size. The principles of path vector routing are shown for a small, simple network in Figure 7, where instead of just communicating hop counts to destinations, the path to this destination is also conveyed. In the example, the simplified routing table of R4 is presented when the network topology data has converged, which shows that the table maps destinations to the best link cost and the respective path. This data would be then sent to neighbors in path vector type update messages which for R4, in the example, would contain destination nodes and the respective link costs, path lengths and list of nodes in the path. It is easy to see that routers can discard incoming path vectors that contain routes which have loops, as they see the whole path and can deduce if a route leads back to the router itself for example. In the case of broken links, the relevant router would then just send path vectors marking the link inaccessible, which would then propagate new paths throughout the network, similarly to distance vector routing update process. It would also be possible to cache multiple routes to the same destination for a quick, smooth change of path in the case network topology changes, while the trade-off being increased overall converge time. With BGP, one difference to the aforementioned simple example is that the link costs are implicitly set to 1 for hop-based routing, where there is a lot of added complexity elsewhere in the protocol to facilitate preferred route selection, among other things.[3, pp. 93–102][13]

An example network structure where BGP is used is illustrated in Figure 8. The

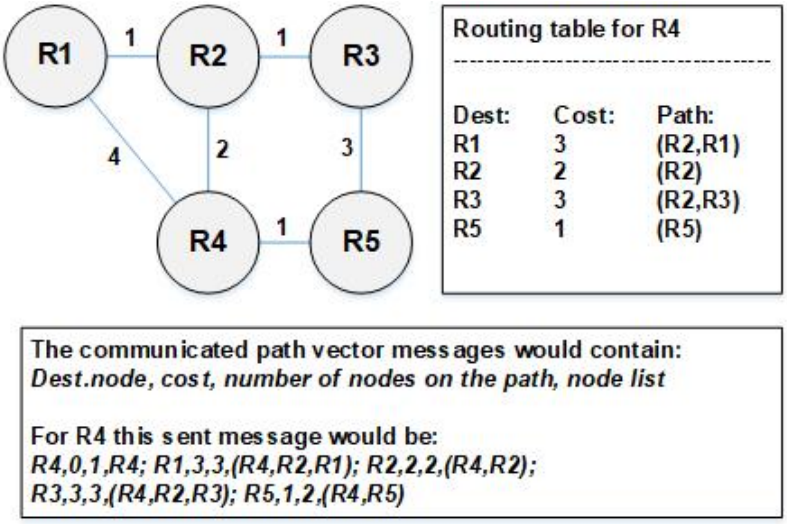

Figure 7: Path vector routing example

basic building blocks of this architecture are the AS entities which are domains with an unique ID value and who are under a common routing policy. Usually each AS is controlled by a single ISP or a group of ISPs that share common goals and financial ties, which then enables the intra-AS routing setups to follow some shared configuration ideas and purpose. The AS identifiers are managed by IANA and are called Autonomous System Numbers (ASN). During the network setup, these numbers are allocated to each AS by the ISP that has control over it. ASNs are 32-bit numbers which denote the node identity, similarly to the router names in Figure 7.[37] These AS nodes then have substructures that can include multiple smaller networks whose IP prefixes are aggregated and advertised to the Internet by the exterior BGP routers (eBGP) that reside on the borders of the AS. If there is just one border eBGP router with the AS, the intra-AS routing could be done purely with Interior Gateway Protocols (IGP) such as OSPF or RIP. With the eBGP advertisements, the AS number respective to the IP prefix is also sent to connect the two and the eBGP routers will then send and receive these messages to have an idea about the IP prefixes and connected ASNs that can be reached in the whole network. In the case of AS having multiple eBGP routers that connect to different networks, the eBGP advertisements are sent through internal BGP (iBGP) routers, as more complex path vector messaging needs to be forwarded through the AS, which cannot be done with the standard IGPs.[3, pp. 239–279][13]

The actual BGP routing process is done by the BGP peers establishing a permanent TCP connection between each other to communicate their capabilities and to exchange IP prefixes and respective paths which are available through them. Each BGP router maintains these TCP connections for each of its peers and regular ”keep alive” messages are sent to notify the peer that the other end is still in operation. If the connection breaks and if re-connection doesn’t succeed, the broken path is then communicated to other peers. In the case of iBGP, there is a problem of loop

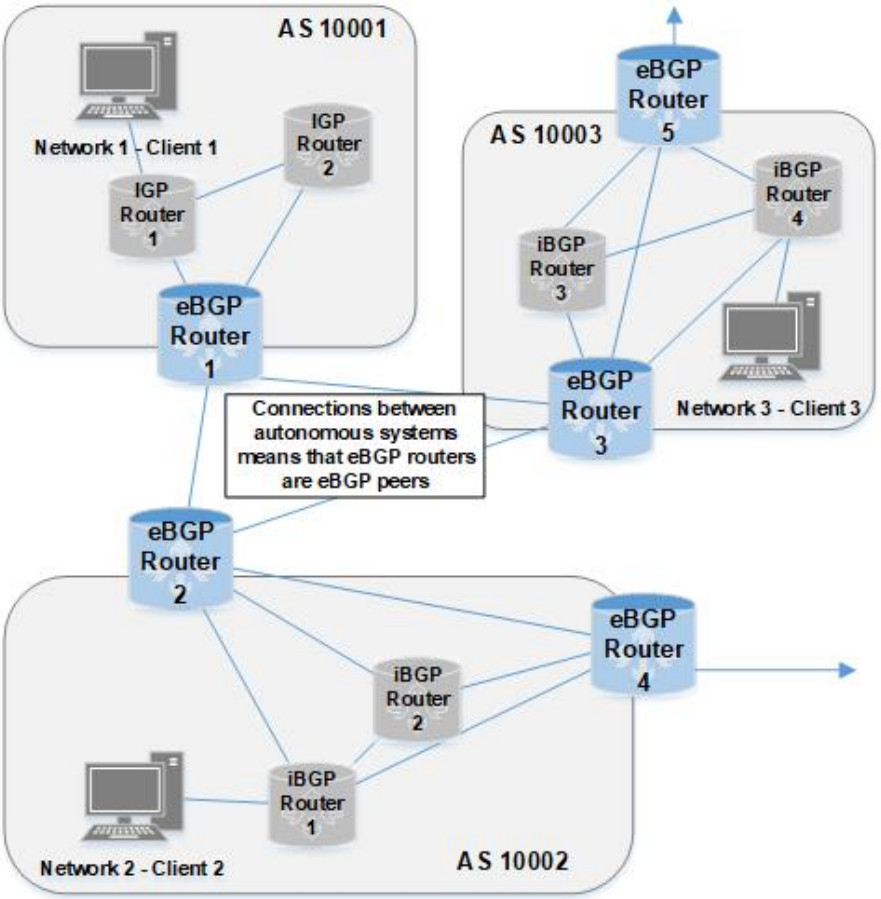

Figure 8: BGP routing and autonomous systems

formation as routers need to omit ASN data from the update messages to differentiate iBGP routing and eBGP routing in a simple manner. This means that iBGP routers cannot advertise iBGP routes to each other, but they still need to be able to advertise eBGP routes between them, which leads to connecting all iBGP routers to each other or using methods like BGP confederations or BGP router reflectors to divide the AS to smaller segments. Overall, even with the iBGP routers residing in the AS, the internal routing is generally done with IGPs which communicate their routing information to the BGP routers if necessary.[3, pp. 239–279][13]

2.2.3 Policy routing, BGP communities and BGP security

The routing with BGP goes beyond just forwarding data traffic to the next hop on the shortest path to the destination. BGP routers maintain separate adjacent Routing Information Base In (adj-RIB-in) data structures for each of their peers, which contain the network topology information received from the respective peer. The routers also maintain a separate local Routing Information Base (loc-RIB) data structure for the actual routing decisions. In addition, the BGP routers keep up adjacent Routing Information Base out (adj-RIB-out) data structures for adjusting

what data they send to the respective peer. The BGP router uses information from adj-RIB-in with its own set of policies and filtering rules to add information to the loc-RIB. This procedure can set preferences for using a particular route over another, filter out IP prefixes which are not to be propagated or filter out a specific AS altogether, if it is a private AS number for example. BGP can also apply these kinds of adjustments and filtering to the outbound traffic, based on the rules on adj-RIB-out, to limit what router actually advertises to its peers. Generally, this concept is called *policy-based routing*, where other factors are taken into account for routing in addition to the IP destination address of a packet. This type of routing is facilitated by BGP sending specific *path attribute* data in the path vector updates to note route preferences and various other metrics.[3, pp. 239–279][13]

One useful path attribute with BGP messaging is the *communities* attribute. Essentially this attribute is a 32-bit value tag, split into two 16-bit values, containing the respective ASN and specific community identifier. This tag can be added to outgoing or incoming IP prefix advertisements, and the BGP router handling these advertisements can then apply rules for the IP prefix based on the rule set matching the community identifier in the tag. An example use of the community tag with a well-known community value of ”NO_EXPORT” (or 0xFFFFFF01 in hexadecimal) is that eBGP routers notice this tag from incoming advertisements from their iBGP peers and know to not advertise IP prefixes with this community to their eBGP peers. An example case of limiting access to specific network within AS to only limited amount of eBGP peers is shown in Figure 9. There IP prefix of network 3, where client 3 resides, is tagged by a specific community tag. The iBGP router 4 would then have rules in place to only advertise this community to eBGP router 3, which in turn knows to advertise the community only to eBGP router 1, which would not advertise the community further on the wider Internet. This enables the client 1 in AS 10001 to access client 3 and network 3 in AS10003, whereas entities in AS10002 cannot do the same, as the respective IP prefixes are not advertised there. The functionality of BGP communities has been expanded since the attribute inception by adding more space to add larger ASNs and type information to the tag, but the principle of utilizing the actual community identifier to enact specific community policies for respective IP prefixes remain [38][39].[40]

Policy-based routing is not limited to BGP, as networked computers generally have routing information in the Forwarding Information Base (FIB) data structures to know where to actually send data. With Linux for example, it is possible to add specific rules such as preferential routes to the FIB so that the routing decision would use other attributes than just the destination address.[41] It is also possible to add routing policies to work with RIP and OSPF, as the routing software usually has support for injecting additional routing rule sets to work in conjunction with the respective protocol. For example, the Quagga routing suite offers a possibility to set some policies based on matching the packet source, which would enable *source-based routing*, where a packet is forwarded to some interface or address based on its IP source address.[42] In regard to policy-based routing, one important use nowadays

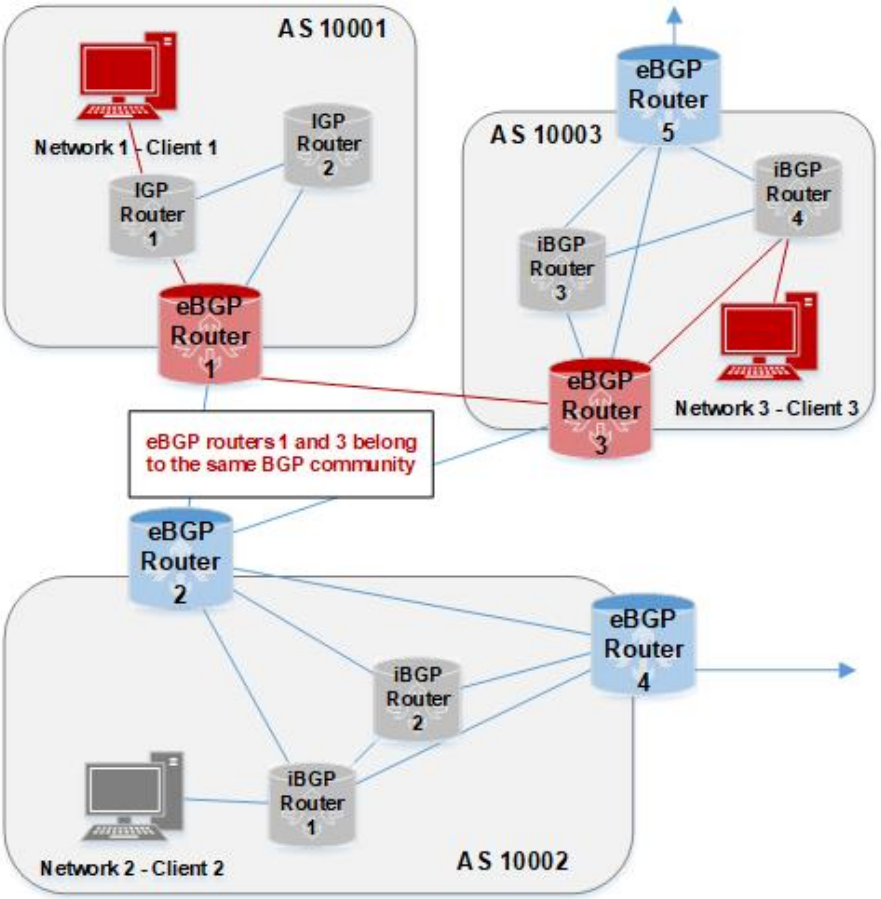

Figure 9: BGP routing communities example

for BGP routers is the creation of Virtual Private Networks (VPNs) mostly for enterprises. VPNs in this context are basically detached, encrypted networks that utilize the BGP's fine-grained controls, where nodes share data only with trusted peers in a manner of their choosing.

With security, BGP may run into the same problems with route advertisements as RIP and OSPF, as the messaging between the BGP peers is done over IP. Advertisements could be wiretapped, and in the worst case, if two BGP peers communicate over the same network segment, malicious entity could possibly hijack the TCP connection between the BGP routers by IP- and ARP spoofing to inject faulty data to the BGP routing information bases. As with IGPs, broken or forged routing tables could create black hole addresses, which lead legitimate traffic to nowhere, or could direct all traffic to one specific route, which could cause DoS-attacks. Additionally, with BGP, undesirable routes could be injected to the AS, replacing legitimate routes, so that outgoing data is forwarded to the wrong destination AS. Overall, issues with BGP routing could cause more damaging effects compared to IGP routing, as compromised eBGP routers specifically affect the whole respective AS. Additionally, BGP networks could have issues with a phenomenon called flapping routes, which is connected to

rapid network topology changes by a link turning on and off in quick succession. At this point, BGP routers would send update messages notifying peers of withdrawing the broken route and then, almost immediately, send another message to announce the route to be on again. These subsequent update messages would then propagate over the BGP network, and in the worst case, cause performance and availability issues with the routers while these problematic message are processed.[3, pp. 239–279]

There are ways to address route flapping and BGP peer connection security, though. BGP routers can enact timers to limit how many update messages are handled from a peer in a limited time to dampen the effect of quick succession of updates. For more reliable BGP peer-to-peer communication, BGPsec can be enabled, where BGP routers could authenticate each other and also use BGPsec to cryptographically authenticate the routes they receive from the path vector messages.[43] One major factor with the general BGP security is the reliance on various operators of BGP routers to act benevolently or competently. BGP routers could be misconfigured so that they are compromised, or a BGP operator that is initially trusted may decide to inject problematic routes to peers or block network access suddenly due to financial or political reasons.[44] It is also possible that BGP routing hardware becomes the bottleneck for some operators, as IPv6 and fracturing of IPv4 network with various private networks has increased the actual FIB size for BGP routers to a point, where older routers cannot cope well with the larger network size.[45]

2.2.4 NAT

As the adoption of IPv6 lags and as the limit of IPv4 addresses has been reached when the size of the Internet has grown especially in India and China, there has been a need to conjure up new IPv4 network address ranges especially for the growing markets. Additionally, there is a need in network security context to detach important web services or network segments in a manner that unwanted traffic from the global Internet cannot easily reach these addresses. This is what *Network Address Translation (NAT)* methods and private IPv4 address ranges are for. Based on Table 1, few ranges such as 10.0.0.0/8 are allocated specifically for private network use, which means that entities such as enterprises and organizations could set up their own private networks, where employees or other stakeholders within these networks could access services there without having to simultaneously cope with disrupting data traffic from outside. There is usually a desire to have some access to the global Internet from the private side, however, as vast array of useful services reside there. This issue has led to wide utilization of NAT services, that act both as network *firewalls* and gateways to other networks. The NAT process can map few or even 1 global IP address of the NAT node to the whole respective private network, which helps with the IPv4 address amount limitations.[6][46]

The concept of the firewall relates to software that runs within a network node, monitors incoming and outgoing traffic of the node's network interfaces and does

some limiting actions to this traffic if certain conditions are fulfilled or rules are to be enforced. Firewall software implementations could be set to work on various TCP/IP model layers, but usually, at the base level, firewalls monitor the IP data traffic in the Internet layer. Example functionality with firewalls include dropping certain protocol data traffic altogether (such as dropping all UDP traffic to the computer), dropping traffic from a specific IP source address range, or blocking specific outgoing traffic to prevent the node from broadcasting some sensitive information. Firewalls can be *stateful* or *stateless*, where stateless versions do actions on a packet-by-packet basis without maintaining information about the data flows and where stateful versions keep up client and connection state information so that, for example, TCP connections through the firewall can be identified and handled properly. As NAT servers are placed on the border of some local network, they usually implement firewall functionality to limit what kind of data is let in through them. It is important to note that some software running a NAT process may not be a firewall itself, but often for convenience, servers running NATs will have firewall processes running in conjunction, which affects how the NAT procedure functions.[11, pp. 763–799][46]

To facilitate communication between the private network space and the public internet, NAT will map private space IP addresses to one public space IP address in a stateful manner, where the private side client connections are differentiated by the NAT assigning them specific port numbers. An example of this procedure with the common many-to-one NAT architecture is shown in Figure 10, where the NAT has one IP address 192.168.0.1 as the gateway for the private side address space 192.168.0.0/16 to which clients can connect to, and another IP address 100.200.1.1 on the public side for Internet connectivity. The client 1, for example, could then access Internet by contacting the NAT, which maps the connection to outside so that the source address of the IP packet is changed to the public NAT IP address and the IP header checksums are also adjusted to match. Data traffic is then sent from the NAT from a specific port, assigned to client 1 here, while the NAT maintains information about the state of the connection such as the original source port number and the mapping of the client IP address to public side port value. There could be a need to do specific Port Address Translation (PAT) methods for how to allocate the outgoing port value but more importantly, TCP connections may require port preservation so that the incoming and outgoing connection ports on the NAT match to ensure that the protocol functions properly. When traffic eventually passes through the NAT, the connection peer on the public Internet then directs data back towards client 1 using the NATs public IP address as a destination. While receiving packets to the aforementioned client-related port on the public side, NAT will then know the respective client, change the IP destination address back to the private address space and forward traffic towards the client.[6][46][47][48]

Even though using NATs can help with the IPv4 address range limitation and with private network security, they have various apparent issues, especially in regard to the end-to-end connectivity. Some protocols such as Session Initiation Protocol (SIP) with Voice over IP (VoiP) may break the protocol layer independence and place IP

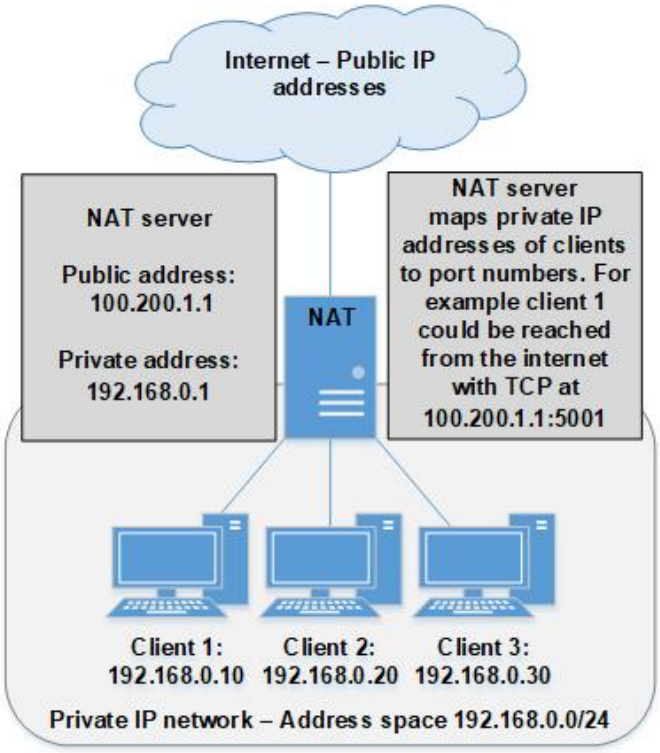

Figure 10: NAT as a connector between different networks

addresses on the application level payload that causes this information to be invalid when NAT changes the IP source addresses, or when IP addresses in some cases would not be unique. It is also problematic when an entity on the outside wants to connect to a client behind a NAT or each connection peer is behind a separate NAT [49]. In a client-server relationship, where only the client resides behind a NAT, the connection procedure works fine as the initiator knows the public IP address of the actual server, but if the roles are reversed, there needs to be specific support for connection establishment built-in to the NAT process, or some NAT traversal methods should be used.[50][51]

If there is a need to include the NAT's public IP address, connected to the NAT client, in the IP payload for some application, Session Traversal Utilities for NAT (STUN) method could be used. With STUN, a client can connect to a specific STUN server on the public internet, which can notify the client about his public NAT IP address and port, as it sees the query coming from this address and port [51]. The issue with all NAT traversal is that NAT implementations and architectures vary and may not be standardized, which means that NATs may change IP addresses and ports for clients when they create subsequent connections and thus the address which the STUN server saw is no longer valid. The option in this case would be to utilize Traversal Using Relays around NAT (TURN) method, where a relay server on the public Internet binds a stable IP address and port for each client after they have connected to the server. The clients are then accessible via these TURN server IP

address and port combinations. The TURN works better with more NAT implementations, but the relaying demands far more resources from the intermediate server compared to the simple query and response process with STUN.[52] For combining both STUN and TURN methods, NAT traversal could be done with Interactive Connectivity Establishment (ICE). With ICE process, peers behind different NATs could gather respective public NAT IP address information to form an address and port candidate lists with STUN and TURN, which is then signaled to the other peer via some unspecified means such as via relay. From this address candidate data, ICE process will then choose the optimal IP and port pairs for the actual data transfer.[53]

Generally, the aforementioned NAT traversal methods enable peer-to-peer communication through NATs in the case of transferring media, using VoIP or utilizing file sharing with BitTorrent protocols and similar peer-to-peer sharing schemes, where there is some initiation from the client behind the NAT. For actually hosting web services in the private network, which are supposed to be accessible from the global Internet, the NAT server itself should have rule sets in place to bind a specific public IP address and port for the private space web service. All incoming traffic to this IP and port would then be sent to the private side service by utilizing address translation and port forwarding. Usually connections through the standard one-to-many NAT are used for a short duration, but hosting services behind the NAT means that there likely needs to be a stable binding, which clients can connect to and which is initialized when the private service starts or by some control signaling either from the private network or from outside.[54]

As NAT masquerades the real IP addresses of clients that reside behind them, people may make false assumptions that just utilizing NATs is enough for network security. Unfortunately, this is often not the case and perhaps the most pressing problem here is that NATs without additional protections offer a good target for DoS- and DDoS-attacks, as attackers could examine which ports are open at the public side of NAT and just send bogus data traffic to these ports to drain NAT server resources. As with routers acting as gateways, bringing down NAT will affect the whole private network behind it. The second matter is that if the private address space client happens to connect to some compromised web service, the attacker could send disrupting data back to the client through the established NAT tunnel. Filtering problematic traffic either by protocol or source address is a good start to help with these security issues, which usually means deploying a proper firewall to work in conjunction with NAT for more reliable and secure network services.

Finally, there is a big question about the role of NATs, when IPv6 deployment advances. Ideally, IPv6 should make NATs obsolete in some sense, because there are plenty of IPv6 addresses for everybody. With the deployment itself, some security issues can come about such as client IPv6 addresses being exposed if security measures are not properly configured when NATs are used for IPv4 addresses with enabling IPv6 connectivity for some dual-stack configurations [55]. There are also several big threats to network security when IPv6 has actually replaced IPv4. If

there is no security-minded network separation, a serious problem will likely be the possibility to utilize vast amounts of IPv6 addresses easily as sources for DDoS-attacks towards a wide range of targets. Another big problem in this case could also stem from the limited security solutions for various, simplistic mobile devices in the global IPv6 Internet. These devices cannot run complex security software due to resource consumption considerations with limited batteries, which could then lead to the devices being easily hacked and exploited for generating DDoS-traffic for example.

2.3 DNS

For most people, the numerical strings of IPv4 addresses and especially IPv6 addresses might be quite difficult to memorize and use in practice, when, for example, one wants to connect to some web service by giving the respective address as an input to the web browser. Additionally, it is difficult to add semantic meaning to these values, which would connect the specific IP address to the purpose or to the operator of the website. These are the main reasons *Domain Name System (DNS)* has been adopted, where textual *domain names*, which are more easily understood by humans, are used in identifying and contacting web services. After the domain name has been given to the query-response type DNS resolving process, it is translated to the IP address of the corresponding web service and the actual data communication can begin where the resolved IP addresses are used. Usually, with web browsers for example, this address translation is done automatically and is hidden from the user, so he only needs to input the domain name to the browser to contact a particular web service.[12][56][57]

2.3.1 Principles of DNS

The domain names, which should be familiar to most Internet users, are text strings which contain text labels separated by dots such as ”www.example.com” or ”suomi100.demo-site.fi”. There are restrictions on these names such as basing the string encoding to American Standard Code for Information Interchange (ASCII) and limiting the character set to include only letters, digits and hyphen. There are also size limits to these names as maximum size per label of 63 octets and the maximum size of the whole string of 253 ASCII characters. The varying labels on the domain name denote different stages of autonomy or authority in the Internet, where these different labels form a tree-like structure which is illustrated in Figure 11. Each label essentially identifies a section, where some managing entity has control over that section and the authority structure below it, all the way to the leaf nodes of the DNS hierarchy tree. The concept of *DNS domain* is often synonymous with a *DNS zone*, although the latter is used more in the technical context when defining how to handle DNS database actions. Usually the authority is delegated from the higher-level DNS zone to the lower-level, so that the lower lever authority has full control over the respective DNS zone. This delegation chain then goes on until the

DNS leaf node is reached.[12][56][57]

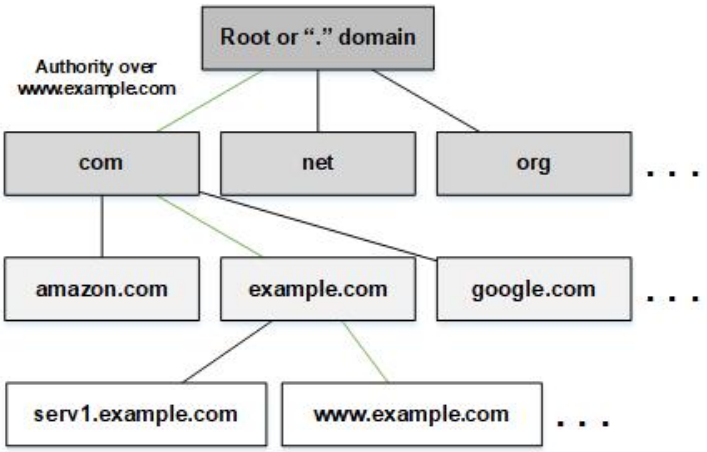

Figure 11: DNS authority structure

With DNS resolution process, the goal is to resolve the IP address connected to the domain that was given as an input. Usually this is a *Fully Qualified Domain Name (FQDN)*, which is a domain name string that has all the labels for respective domain name without omissions. The first step of the resolving process would lead to asking the highest authority, or the root authority, on the tree-structure about the FQDN. The root authority is commonly marked as a dot, which is included in the FQDN in the actual querying process, but this is usually appended automatically to outgoing queries by the querying application, so it doesn't confuse users. For example the actual transmitted DNS query for domain "www.google.com" would be "www.google.com." as it should initially go to the root, even if users usually give only the former as an input. In the aforementioned Figure 11, for resolving IP address to domain "www.example.com", the root has knowledge about Top Level Domains (TLD) such as "com" or "net" and it can then direct the query to this specific domain authority. The TLD authority can then direct the query to second level authorities, where companies, organizations and even normal Internet users can purchase domain names to their web services. The domain name authority chain can then go on if the second level domain names need to have further subdomains, but eventually there is a final stopping point, where the authoritative entity has the mapping of the domain name to the relevant IP address, which is then conveyed to the DNS query originator.[12][56][57]

The entities who actually respond to DNS queries are called *DNS servers*, whereas the querying clients are called *DNS resolvers*. In the common use case of resolving a DNS domain name with a web browser, the DNS resolver is usually an ancillary process for the browser on the client's computer that contacts DNS servers and then responds to the browser with the resolved IP address, if the DNS query was a success. DNS servers maintain a distributed database about the mapping of domain names to service IP addresses, where a server can be marked to be *authoritative* for a specific domain or

DNS zone, which is a more common denominator, when DNS database structure is discussed. This basically means that the server can actually respond to queries about domain names within its zone with an IP address, as it has this information in its database. The role then for non-authoritative DNS servers is to direct queries further in the DNS hierarchy towards this authoritative DNS server. The querying processes can be either *iterative* or *recursive* which are compared in Figure 12. With recursive process, the DNS resolver within the client's computer queries the local DNS server for some address that then queries entities on various level of the DNS authority hierarchy, layer by layer, until it is directed to the actual authoritative DNS server related to the FQDN of the query. It then receives the answer from this server which can be relayed to the client. In the iterative process, the local DNS server will respond to the query by relaying the address of the root DNS server, and the layer by layer querying process is done by the client's DNS resolver instead, until it receives the actual answer from the same place as it was retrieved with recursive query. It is good to understand that recursive queries are more taxing for the DNS system, so the use of recursion is often limited. In most cases DNS client would be allocated a "local" DNS server by his ISP (the leftmost DNS server in Figure 12), which is likely the only DNS server that would do recursive queries on behalf of the client. These imposed limitations make the whole DNS system more stable, though, as root servers can accept aggregated queries from trusted lower-level DNS servers and can filter out iterative queries from unknown sources for example. With problematic DNS queries, it is important to note that recursive queries going through the "local" DNS server will not convey the source address of the original query. This can make it more difficult to find the real source of troublesome queries deeper in the DNS system.[12][56][57][58]

The root domain and TLDs are under the control of Internet Corporation for Assigned Names and Numbers (ICANN), and the domain name always has a TLD label which is the rightmost text tag in the FQDN. TLD name list is relatively rigid and names usually link to either somewhat vague purpose of the domain such as "com" for commercial websites and "org" for organizations, or to country code denoting the actual location of the domain such as "fi", "se" or "de" for example (for Finland, Sweden and Germany respectively). On the second domain hierarchy level and beyond, names can be based on company names, products, location names, etc., where the domain name choice is relatively free. Obviously, most web sites desire domain names that can be semantically connected to their actual purpose. The distribution and registration of these names below the TLDs are done by domain name registrar entities which have received authority for this from ICANN. Usually, the final, leftmost identifying label of a FQDN, which actually ends up pointing to some network node, is called a *hostname*, and in many cases, network users create hostnames for some new service, which are then added to work under a specific domain. As a simple example, when a client wants to add a new service to the Internet, which should be accessible by FQDN "service1.example.com", he can contact the domain name registrar responsible for the "example.com" domain, who can then add the label "service1" (hostname) to be a part of the DNS database, assuming that the client has provided an IP address to be connected to the new

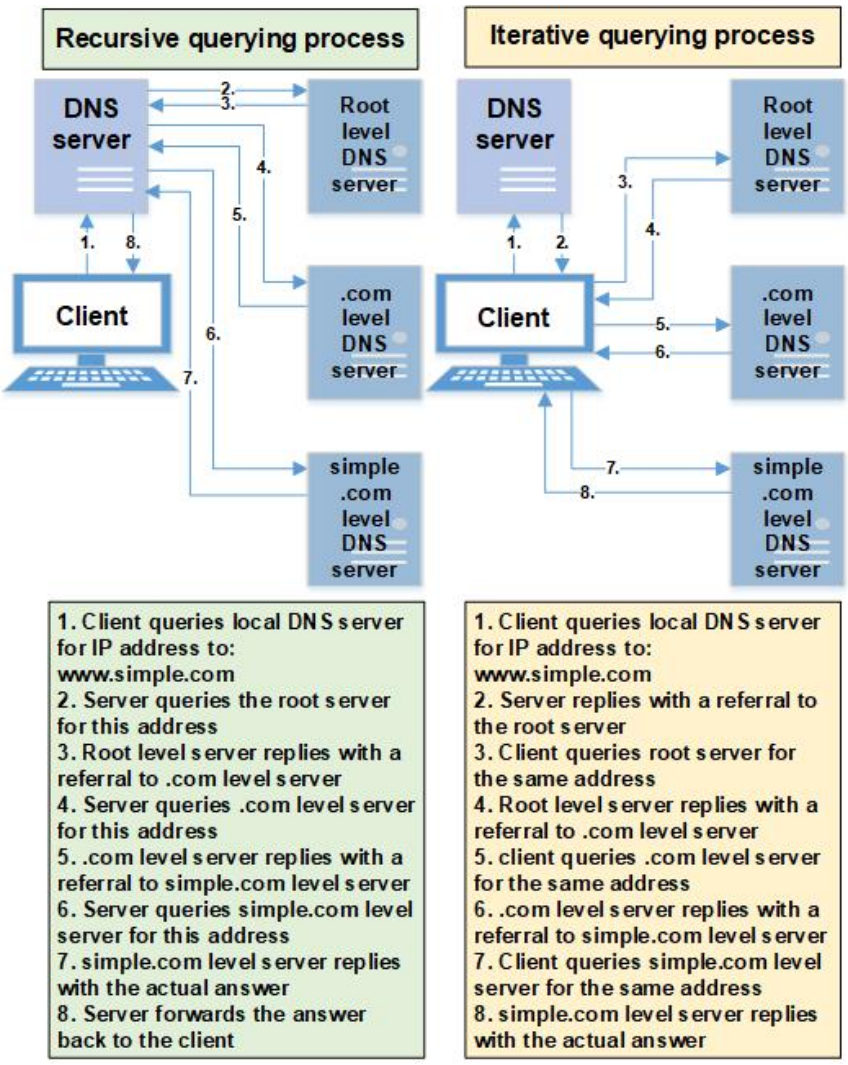

Figure 12: DNS querying processes

domain name. This IP address can then be retrieved by querying the authoritative DNS server of the ”example.com” domain with FQDN ”service1.example.com”.[56][59]

Additional important features of the DNS system include caching of answers and ability to do reverse address lookups. DNS resolving process could include multiple consecutive queries to different level DNS servers, which add delays to the whole process and can demand lot of resources from the higher-level servers. The DNS database contains information on how long various answers should be cached further on the DNS system, which is then noted by the intermediate DNS server when it receives an answer for the first query for the respective domain. In Figure 12, with recursive query, caching the answer in the local DNS server would save a lot of time and effort if subsequent queries for the same FQDN are made by different clients. DNS answer caching is actually the main reason to use recursive queries, as various intermediate DNS servers could have cached answers at least for reaching the often

used ”com” TLD server, which eliminates the need to visit the root. In regard to reverse lookups, DNS system supports query resolution based on input IP addresses, where the FQDN connected to the given IP address is returned. Usually the ISPs which control the respective IP ranges will delegate the reverse DNS lookups to the relevant entities in the DNS system. Note that while reverse lookups can be useful, they are just an optional feature of the DNS system and an arbitrary DNS server on the Internet may not support them.[12][56][57]

2.3.2 DNS messages and resource records

The communication in the query-answer type DNS system uses application level DNS messages that contain data objects called *Resource Records (RR)*. These messages are sent usually over UDP, where DNS servers generally listen on port 53 for incoming queries. The DNS message format is presented in Figure 13, where the message contains a header section and a variable number of RRs. The maximum size for UDP DNS payload is 512 bytes, where this is added to the header and the total size is assumed to be below 576 bytes in size, which then enables servers to reassemble fractured UDP messages, as per the IPv4 specification [4]. It is also possible to make DNS queries over TCP, which enables the transmission of larger messages, although this can be done with UDP too with DNS extensions that specify additional functionality to the DNS system [60]. As for the purpose for the specific fields in the DNS message header above the RR data, they are explained below:

- DNS message Identifier – This identifies the response message for the DNS client as the DNS query and the respective response should have the same identifier value
- Query or response-flag – Notes if the message is a query or an response as these two are the basic types of the DNS message
- Operation code – Denotes additional information about the purpose of the message such as marking it to be a standard query/answer type, a server status query, a notify or an update message, where the last 2 relate to updating the DNS server address database dynamically
- Authoritative answer-flag – Marks the answer to be authoritative which means that it came from the entity that had authority over the DNS zone respective to the queried FQDN
- Truncated-flag – Marks that the message was too long to fit in the 512 byte payload, which usually means that the message should be sent over TCP, which in turn could be used as a way to make sure that the querying DNS resolver doesn’t use forged source address
- Recursion desired-flag – Asks for the query to be resolved recursively if possible so that the DNS server in question would do further necessary queries on behalf of the client

- Recursion available-flag – Marks recursion to be available or not on the queried DNS server
- Status – Used in DNS responses to mark if the query was answered successfully or to note what error occurred if something went wrong; example error codes include format error where the original query message was malformed or name error where the queried FQDN was not found in the domain
- The count-fields – These denote the count of each specific RR type in the message RR payload

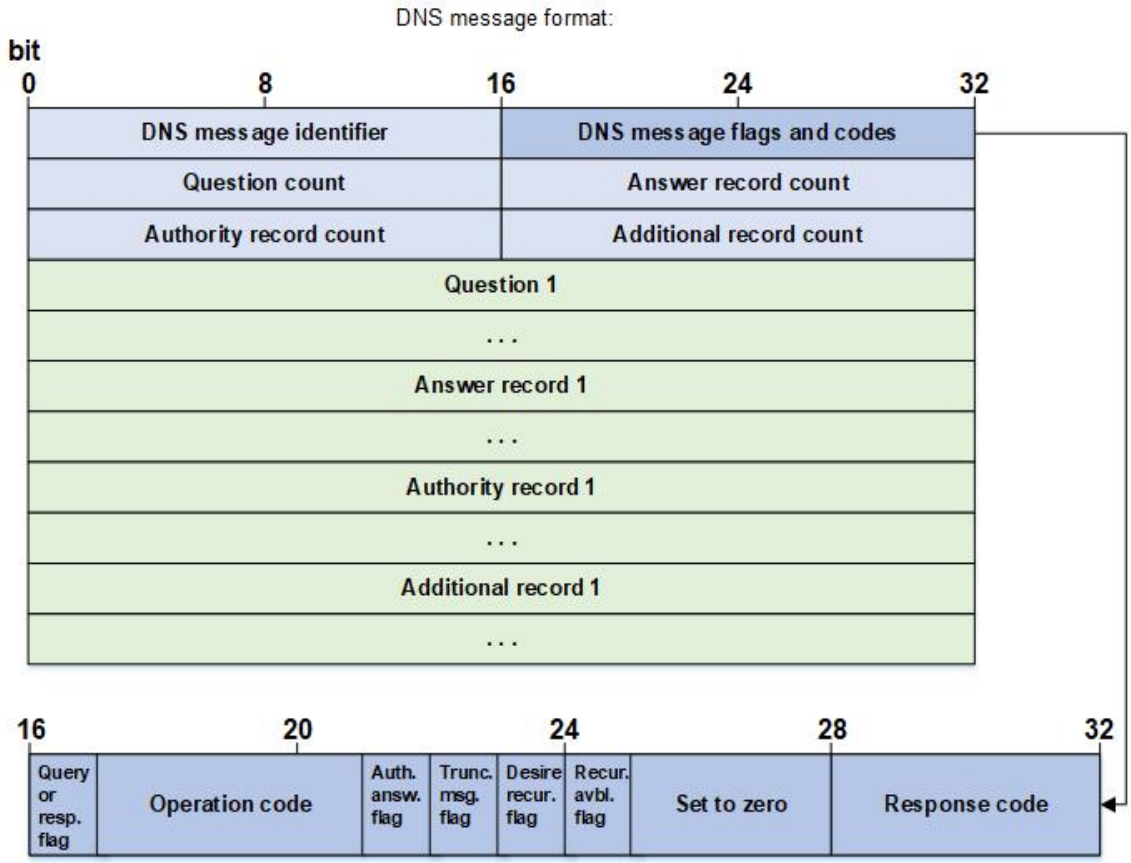

Figure 13: DNS message header

As was implied on the listing, there are few functions available using DNS messages, which go beyond the query and response-scheme. Originally, the data on DNS servers was thought to be relatively static, where the frequency of change could be handled by just adjusting databases when servers were shut down for maintenance. Later it became apparent that there is a need also for dynamic updates, as the pace of change has increased in the Internet. The growing DNS system has also led to incorporating multiple DNS servers on 1 authoritative DNS zone with the master and slave structure, where there are backup servers available if the master DNS server goes down. With the notify operation code, the master DNS server can notify

and adjust slave servers in the case database changes, whereas the update messages can be used with Dynamic Domain Name System (DDNS), where the DNS server databases could be changed quickly and in a lightweight manner, preferably using automated management clients [61][62].[12][57][63][64]

The DNS domain name and IP address connections are conveyed with DNS RRs, where these records for a particular DNS zone are contained in the DNS server database file. In the DNS query process, the DNS server attaches the relevant RR information from its database to the answer section of the response and possibly to the authority and additional sections as well. The DNS resolver can then parse the relevant data from these records on the DNS reply message to initiate the IP connection. The common DNS RR types are shown in Table 3, where Domain Name System Security (DNSSEC) note implies that records are connected to optional DNS security functions. The actual IP addresses are stored in A and AAAA RRs for IPv4 and IPv6 respectively where, in most cases, this record is returned when a DNS query comes in for the server's DNS zone. For more advanced use cases, the *Canonical Name (CNAME)*, Mail Exchange (MX) and Name Server (NS) records are redirecting records, where they denote aliases, name servers and mail agents respectively for the DNS zone in question. These types of records then eventually lead to A or AAAA RRs, where the actual IP addresses for these entities are stored. How RRs are managed by the server is illustrated in Program Code Snippet 1, which shows an example DNS database file for the BIND software that is the most popular DNS server software currently in use on the Internet. As can be seen from the code, the A records contain the respective FQDN on the left, followed by the class of the record which is usually IN for Internet, the record type and lastly the respective IP address mapped to the FQDN. CNAME records are similar, although they contain the FQDN of the alias, which should be used in a consecutive query, instead of the actual IP address.[64][65][66]

In addition to storing the address mappings, the RR database files contain an important Start Of Authority (SOA) record for the DNS zone at the beginning, which is also shown at the start of the Program Code Snippet 1. This record holds configuration and timer information for the DNS process. The master DNS server is denoted after the SOA RR type, where the serial value denotes the version of the file. The refresh, retry and expire relate to the behavior of slave DNS servers on how often they should query the master for zone updates and how long the records they maintain are valid without updated data. Lastly, the ttl-field denotes time-to-live, where this describes how long the answers from this DNS zone should be cached further in the DNS chain, although this value could also be configured to each record individually.[12][66]

One important feature for extending the DNS functionality is the OPT type (pseudo) resource record which showcases records that are not stored on the resource record databases but are added to the DNS messages for system signaling and configuration purposes [60]. Expanding DNS also connects to enacting security measures with

Table 3: DNS resource record types

RR type	**Purpose**
A	Stores the 32-bit IPv4 address, usually for some hostname
AAAA	Stores the 128-bit IPv6 address, usually for some hostname
CNAME	Alias to a hostname; the DNS lookup for IP address will continue using the alias
DNSKEY	Public key data for validating RRSIG record signatures (DNSSEC)
DS	For identifying the signing key for a delegated zone (DNSSEC)
MX	Mail transfer agent listing with priority information
NS	Authoritative name server address for the DNS zone
NSEC	Link to a next record name to confirm the non-existence of a record if necessary (DNSSEC)
OPT	Pseudo-record for use with DNS extensions
PTR	Pointer to a CNAME, where this name is returned but the DNS resolving process doesn't continue as it is usually the case with normal CNAME
RRSIG	Secured record set signature (DNSSEC)
SOA	Marks the start of authority record and specifies configuration and timing information for the DNS zone
TXT	For arbitrary textual information; nowadays mainly for transmitting machine-readable data for various additional DNS-related protocols

DNS in the case of authenticating and validating resource record data. The security processes thus can use further, additional resource records which is the case with DNSSEC.[67][68]

2.3.3 DNS extensions and security considerations

The mechanism to extend DNS beyond adjusting the original DNS header setup is called *DNS extensions (EDNS)*, which can be referred also as EDNS0 for the basic version 0 of this mechanism. One of the main purposes of implementing EDNS has been adding security features to the DNS system but there are other uses for DNS extensions too, such as forwarding client subnet data towards the authoritative DNS servers in the recursive DNS process with *EDNS Client Subnet (ECS)*. This information could be used by the leaf DNS server, for example, to choose optimal IP addresses for resolving the FQDN, based on the client's geographical location, which

Code Snippet 1 - Example DNS database file used by BIND

```
@    IN SOA ns1.example.com. admin.example.com. (
     29          ; serial
     3600        ; refresh
     1800        ; retry
     604800      ; expire
     600 )       ; ttl
; name servers - NS records
IN      NS      ns1.example.com.
; name servers (A records)
ns1.example.com.          IN      A       11.22.33.44
; 11.22.33.0/24 - A and CNAME records
host1.example.com.        IN      A       11.22.33.51
host2.example.com.        IN      A       11.22.33.61
host3.example.com.        IN      CNAME   host1.example.com.
```

could be deciphered from his subnet address. The EDNS pseudo resource records contain a fixed part at the start which denotes the existence of EDNS RR type with value 41 and the EDNS version, after which there is a variable amount of actual options. Example of this specific option data is shown in Figure 14 which describes the ECS option format, where at the beginning, there is a code for depicting the option type, the length of the option record and after that, data relevant to the actual use of the option. This basic structure is common for all types of EDNS options, and in this case, the option data payload includes the client subnet address range, adjusted by the source and scope prefix lengths to manage the accuracy of the forwarded range information.[60][69]

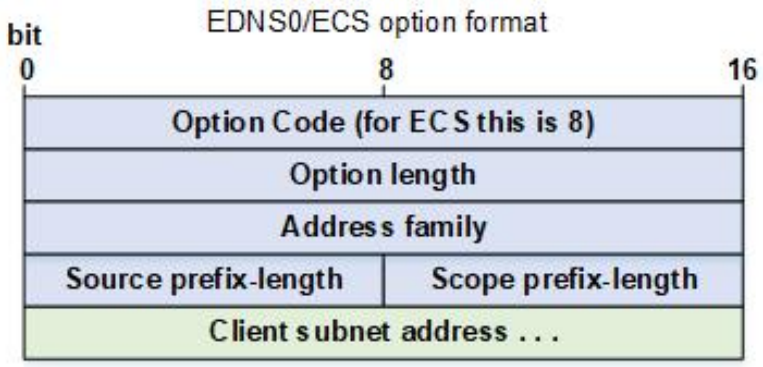

Figure 14: DNS ECS option resource record

As DNS is an application layer protocol which works usually over UDP and IP, it has to take into account packet capture problems that come with insecure UDP/IP data traffic. The main issue with DNS security is that MitM-type attacks could be done by malicious entities intercepting DNS queries either between recursive DNS server

and authoritative DNS server or between the client's DNS resolver and the local DNS server. The attacker could then send forged information back where false addresses are given to the DNS resolvers, which would direct the client to some hostile IP address. Additionally, MitM-type attacks can be used to do DNS cache poisoning, especially in the recursive DNS process, where false information is injected to the DNS server or resolver cache, where faulty, cached address continues to direct to the wrong address. Cache poisoning can lead to name chaining attacks, where a CNAME record for example is used to redirect clients to resolve some arbitrary FQDN.[70]

In order to solve the aforementioned DNS security problems, DNSSEC was developed for validating DNS RRs by attaching a digital, cryptographic signature to the records of a particular DNS zone. This idea links to using public and private cryptographic security keys which are elaborated further in the next chapter, but for clarification, the relevant DNSSEC resource records were shown in Table 3, marked with the DNSSEC note. In the DNSSEC hierarchy with a chain of trust and with using EDNS, the root zone's Delegation Signer (DS) record can be used to verify the DS and DNSKEY of the TLD server. Then, the DS of the TLD in turn can be used to verify the keys in a connected lower-level domain and so on. The resource record validity of an authoritative domain can finally be verified by checking the Resource Record Signature (RRSIG) of the RR set with the DNSKEY of the DNS zone, where the trust for this DNSKEY was passed down from the root. The existence or non-existence of a particular RR can also be verified with the Next Secure Record (NSEC), to spot additional, falsified RRs. The DNS system integrity can be maintained with DNSSEC if it is deployed effectively, but there are still other cybersecurity issues with DNS such as DNS servers being susceptible to DoS- and DDoS-attacks. It is easy to see that spoofed UDP queries could be sent to critical DNS servers to drain computing resources and cause delays, if not total DNS system shutdown, which can certainly be a problem for lower-level DNS servers. At least for the higher levels on the DNS hierarchy, *anycast* IP addressing is employed for load distribution, where there are essentially multiple DNS servers behind a single IP address. Using anycast, traffic is directed automatically towards the closest DNS server behind an anycast IP address, from the client's point-of-view, using the standard Internet routing. This, at least in principle, makes widespread DDoS-attacks difficult to implement versus singular, critical DNS servers, but deploying anycast in a larger scale can be very difficult due to routing complexity from simultaneous use of many identical IP addresses [71].[68][70][72][73]

3 Internet service security

In the context of network service security, there are three main concepts stemming from computer security field, which help define the parameters for a secure web service: confidentiality, integrity and availability. Confidentiality generally means access control, where access for the service is only granted to entities that have the proper credentials which are authenticated usually during establishing the communication session. Confidentiality also goes beyond having just rights to access some service, as it is often important to protect sensitive data traffic going between authenticated clients and some secure web service. In these cases, data encryption can be utilized, where only entities with credentials can decipher the sensitive messages being sent.[1]

Often, it is also important to ensure that the received data in telecommunication is not malformed or tampered with and is from the correct sender, all of which link to the integrity principle. Methods for helping with these issues usually include utilizing checksums and length fields in protocol headers, as these values can be used to validate data packet contents to some degree. More robust methods can utilize calculating a *hash value* from the data packet contents which is a unique, relatively short, one-way, textual representation of the content, received from a mathematical hashing algorithm with the original data as input. The differences on data packet contents could then be noted by the receiver when he compares the hash value in the packet header, for example, to the hash value calculated for the arrived packet. For resolving errors in the content, the process usually just involves a request to re-send the data if specific error correction methods are not in use. For the problem of validating the sender for some arbitrary data traffic, the common solution in the Internet is the use of *Public Key Infrastructure (PKI)* that relies on *public key-encryption*. With PKI, network entities have a *private security key* and a *public security key* pair, where public keys can be distributed to everybody. The relevant, common use case of public key encryption is that the data signed with a private key can be validated by using the corresponding public key. In many cases PKI is also used to facilitate user authentication and data encryption, where a common example would be a client connecting to some secure web service, such as a bank website, with HTTPS which utilizes TLS and PKI.[1][74]

Lastly, the availability concept in the cybersecurity context means that a service should be available to use when needed. This can relate to maintaining and updating the service hardware and software to a point, where the probability to encounter sudden system downtime due to hardware malfunctions or software bugs is very low. Additionally, backup systems could be installed to take over if sudden service breakage occurs. In this thesis, the focus, however, is on observing the proactive and reactive measures that can be taken against outside actions that try to affect service availability. Usually this means that malicious entities try to inject lots of illegitimate data traffic to network systems, with DoS- and DDoS-attacks, to use up system's processing resources, memory and network bandwidth. These actions would then make the system to be unavailable for actual users, as it may shut down due to

software or hardware failure in the face of abnormal amount of sudden data traffic. Even if shutdown is avoided, the system may be too preoccupied by processing the bogus traffic and would then reject most of the legitimate clients. Many times, efficient traffic filtering methods such as utilizing firewalls can be used to reject the attack data flood, especially if small amounts of IP source addresses or specific port for the hostile traffic can be pinpointed. As address spoofing is quite easy to do on the Internet, source-based filtering may not be as effective, though, so DoS- and especially DDoS-attacks can still pose a major threat to web service security.[1]

This chapter first discusses the user authentication data encryption basics and briefly introduces PKI and TLS, which are the cornerstones of modern Internet security, where data goes over insecure communication channels. More detailed description of PKI and TLS is included in Appendix B. After this initial part, the threat landscape in the web service availability context is presented, where various DoS- and DDoS-attack types are discussed. Following this, there is a section about the defense measures that can be taken against these attacks. In the last part of the chapter, the Linux OS network security functions and the Realm Gateway software are presented, as they are an important part of the actual testing procedures in this thesis.

3.1 Encryption, authentication and PKI

On the very basic level, secure Internet communication between network peers would require some verifying methods, so that only the intended recipient would be able to utilize the communicated messages. First concept that links to this is the *symmetric encryption* scheme, where the first task is to create an encryption key, which is usually some randomized text of reasonable length. Using this key and the to-be-encrypted data as an input, chosen encryption algorithm then gives out encrypted data that can be sent to the other peer. In symmetric encryption, it is assumed that the recipient has access to the same encryption key which he can then use to decrypt the message with the corresponding decryption algorithm, which takes the encryption key and encrypted message as an input and gives out the actual message. For the actual encryption procedure, various methods are available such as commonly used block ciphers, where a deterministic transformation algorithm is applied to the input data in a block-by-block manner with the blocks being a sequence of bits of pre-set length. In a more formal manner and to present an example, generalized block cipher encryption function is

$$E_K(P) \equiv E(K, P) \colon \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n, \tag{1}$$

where E is the encryption algorithm with some arbitrary plaintext P and the encryption key K as an input. The parameters k and n denote the key size and the block size in bits respectively, where an encryption result for n-sized block of plaintext is then n-sized block of encrypted bits. The decryption is then done with the inverse function

$$E_K^{-1}(C) \equiv D_K(C) = D(K, C) \colon \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n, \tag{2}$$

where D is the decryption function which takes the encryption key K and the encrypted text C as an input and calculates out the original plaintext for each block. In respect to K,

$$\forall K : D_K(E_K(P)) = P, \tag{3}$$

which means that with arbitrary key and arbitrary input text, the encryption and subsequent decryption of this text will be possible, if the K is the same for both procedures.[75] Block cipher algorithms commonly in use nowadays include Advanced Encryption Standard (AES) and Triple Data Encryption Standard (3DES) which is based on using Data Encryption Standard (DES) algorithm three times over the input. Both of these algorithms base the encryption on enacting multiple rounds of permutations such as bitwise operations or value re-arrangement on the plaintext blocks to produce a pseudo-random permutation from this text based on the encryption key.[76][77]

In addition to encrypting telecommunication, the issue of net entity authentication is also important. This is more relevant to server-client type web services, where a service such as web bank maintains a database of authorized client identities. The client would then connect to this service, and during the connection setup he transmits his user identity and the respective password to the service which would then compare this to its user-password database to enable authorization. One major issue with this very basic scheme is that if specific traffic sending the username and password is eavesdropped, the attacker could then utilize the same credentials later. One way to combat this problem is for the server to send a additional text string or salt value to the client, where client will calculate a hash value with a one-way hash function using the combination of his password and the given salt as an input. This hashed value is then transmitted to the server that checks this value for the given username, as it can calculate the same hash result from the respective password and the salt it sends to the client. The attacker would not then be able to retrieve the actual passwords from these messages, as the point of one-way hash functions is that one cannot calculate the input easily from the output. Authentication method, where the server will provide some parameters to the client to be processed and to be used in hashing the password information before sending it to the server, is called challenge-response identification, from which a Challenge-Handshake Authentication Protocol (CHAP) is a simple example [78]. With authentication, it is also good to note that nowadays the password databases usually contain hashes of the passwords, which means that if the database is compromised, it doesn't automatically lead to unauthorized access.[75, pp. 321–420]

Two security challenges that relate to symmetric encryption and user authentication in client-server relationship must be noted. The first problem is transmitting the symmetric encryption keys to communication peers over the Internet. There could be a separate channel for sending this information such as the peer calling the other by telephone and then telling him the key for example, but these processes become more burdensome if the peer is distant and doesn't have a secure contact line. Additionally, there can be trust issues, when a client has to be sure that the server he contacts, when giving out his credentials, is the real deal and not some malicious

entity masquerading to be the server. This is where the *Public Key Infrastructure (PKI)* using *public-key cryptography* comes in.

The basis of PKI is public-key cryptography which is an *asymmetric encryption* scheme. In contrast to symmetric cryptography, two different encryption keys are used here: *the public encryption key* and *the private encryption key*. The idea is that data to be transmitted can be encrypted with the public key and then this data can only be decrypted with the private key. The creator of the key pair can then publish the public key to be used by anybody, who can then send encrypted messages to the key creator which only he can decrypt. Additionally, the key pair creator can sign messages using his private key in combination with the text payload, where the receiver can then verify this message using the public key to ensure that it was signed by the corresponding private key and it was not changed during transit. The major benefit of public-key cryptography is that web services or even normal web users can distribute their public keys freely, which can then be used to ascertain their identity as they can send signed messages during the connection establishment. Example of this would be authentication where a client tries to connect to some web server who, in turn, can send identity assurances with a signed message before the client submits his credentials.[75, pp. 283–312]

With PKI, the difficulty in solving the private key when public key is known can be based on well-known mathematical challenges. For example, using powers and modulo with the private key in the message signing and hashing process can be utilized, as factorizing prime numbers is problematic especially in a short time period. If the cryptographic parameters are chosen properly, the base of PKI security is quite solid at least with up-to-date security algorithms. Additionally, PKI requires infrastructure and trusted entities to distribute the public keys, mainly to make sure that the given public key for some web service can be linked to the actual web service and is not forged. The PKI is then utilized in conjunction with TLS when HTTPS connections are made. With TLS, a secure connection is created with the use of session-specific encryption keys that are partly based on the public and private keys of the connection peers, which enables encrypted network connections for validated services with the possibility to use client authentication. More detailed description on how PKI, the related key distribution and TLS works is presented in Appendix B.[75, pp. 283–376]

3.2 Relying on PKI

The security of the encryption and key exchange schemes in PKI varies depending on if the utilized algorithms are up to date and the input parameters such as the given encryption key are of sufficient length. A clear boon for PKI with modern encryption is that the brute force approach in decrypting TLS session messages for example is computationally very intensive. The fastest known algorithm for factoring large integers, General Number Field Sieve (GNFS), which could be used to solve the

private keys with common PKI algorithms, among other things, has the complexity

$$C(n) = O\{\exp[c(\log n)^{1/3}](\log\log n)^{2/3}\}, \tag{4}$$

where O denotes the big O-notation that describes how the given function behaves as it reaches a certain limit or infinity and where the constant c depends on the exact version of the GNFS (for "general" cases it is 64 divided by 9 to the power of one third, which is roughly 1,92).[79] The key thing to note here is that the time complexity or computational complexity of GNFS is

$$T_{Example}(n) \approx 2^{n^{1/3}}, \tag{5}$$

where $T_{example}(n)$ describes how the number of required computations increases for solving the particular security key, when the input size, or key length n in this case, for the security algorithm increases. The time complexity here is sub-exponential which means that the increase is somewhere between exponential growth and quadratic growth, which is also illustrated by Figure 15 showing how fast the resource demand grows in logarithmic scale when process types of different time complexities are compared. From this figure, it can be seen that the run times for GNFS will start to rise faster than the cubic function when the input goes beyond 100000.[80][81]

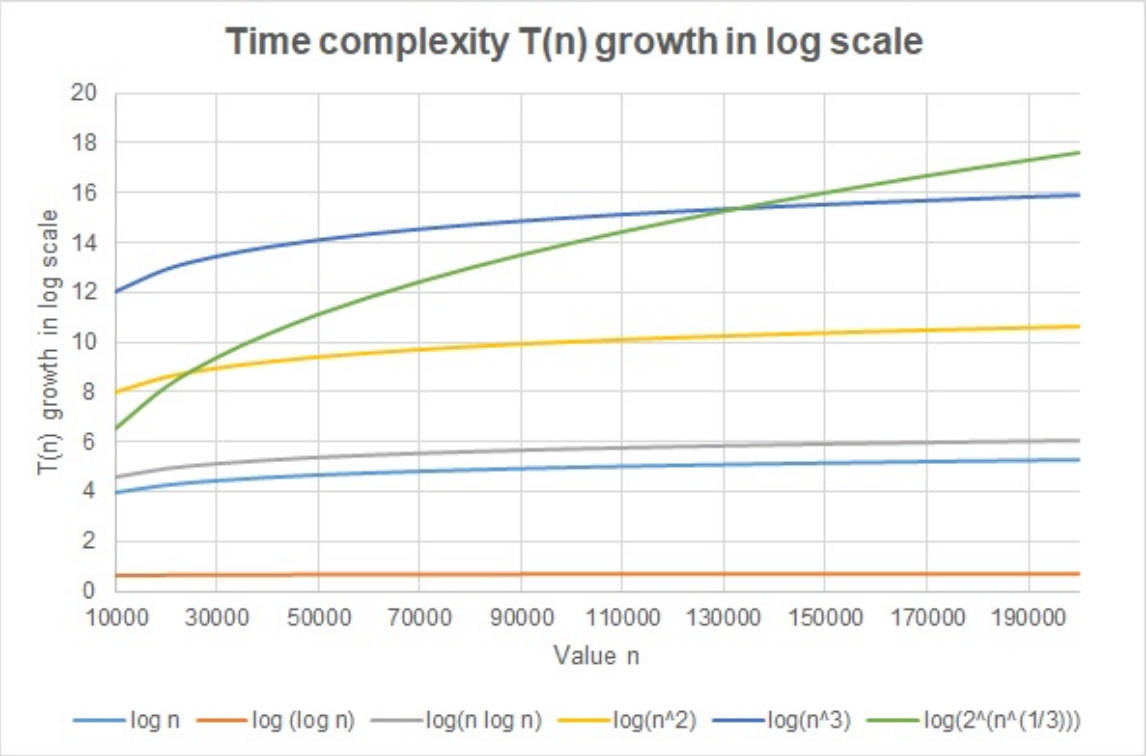

Figure 15: Time complexity growth comparison

As Figure 15 implies, solving private key components in the PKI context becomes very burdensome when the input grows more and more, even by the state-of-the-art

factorization methods. For example, solving the private key for Rivest-Shamir-Adleman (RSA) algorithm, which is discussed in Appendix B, with properly chosen input parameters and with 768-bit modulus could take over 2000 years of computing time from a single 2 GHz processor in 2009 [82]. This means that it is not feasible to crack session based private keys that are used only for a short time during the TLS handshake, if the encryption parameters are large enough. The same resiliency holds for modern symmetric encryption schemes such as AES with 256-bit keys too. Cracking this AES encryption with 2 to the power of 256 different key possibilities just with brute force approach, with a 2 GHz processor and with one key checked per clock cycle to simplify, would take 10 to the power of 60 years in order to exhaust the key space. With strong symmetrical and asymmetrical encryption schemes, usually the problems with data confidentiality and integrity are caused by human error, software bugs and design faults, and built-in weaknesses in certain mathematical algorithms in use for the encryption and hashing processes.[1]

If inherent security algorithm problems are scrutinized more closely, one of the more apparent issue is the existence of hash collisions with hashing functions. As hashing ideally creates a unique, often condensed, output from arbitrary input, it is possible that two different inputs would create the same hash, especially if the hash length is limited. Exploiting hash collisions would make it possible to forge hashed signatures even if the forger doesn't have access to the actual private security key. An example case of this kind of vulnerability is the Message-Digest 5 (MD5) algorithm hash collision issue, where the expert consensus is to avoid utilizing this common protocol for any serious use-cases [83]. Often protocol design weaknesses can be fixed by adjusting the security suites to use other, more recent hashing or encryption algorithms as a base, as there are many options available. With TLS, the challenge usually lies on the inter-operability of multiple different components in the TLS security suite rather than on the security of the best pieces there. Software configuration faults and problematic 3rd party programs may cause the TLS session to become compromised, where one example of this kind of issue with older TLS versions is the downgrade attack, where MitM-attacker could inject false initial TLS cipher suite parameters to the server on behalf of the client to make the session use less secure. Another example issue of similar ilk is the truncation attack, where attacker can inject un-encrypted SYN-FIN TCP messages towards the server to terminate the TLS session during the client's TLS log-off process in order to keep the actual TLS connection up without the client's knowledge. This would enable the attacker to later access the service using the client's account.[84][85]

Provided that the cipher suites and hash functions used with PKI are updated to be secure such as using AES for encryption, Secure Hash Algorithms version 2 or 3 (SHA-2, SHA-3) for hashing due to them having no apparent hash collision faults, and that the key generation parameters are chosen properly, PKI can form a good base for ensuring data confidentiality and integrity [86]. This comes at some cost, though, as encryption and decryption of data demand more processing resources compared to handling normal, un-encrypted data. Additionally, transmitting digital signatures

attached to messages will cause network traffic overhead as lengthier signatures can be quite noticeable compared to UDP and IPv4 packet sizes for example. When one examines attacks versus PKI and TLS, they often include exploitation of short-lived software bugs or deprecated software that is still in use by neglect and this may require significant skill, research and preparation from the attackers. In contrast to this, there is unfortunately a simpler and more straightforward way to affect web service security, which is to utilize the denial of service in the availability context. The issue here is that generally most web services are supposed to serve the public somewhat leniently and not discriminate against clients at least initially. On the other hand, it takes far less resources for a client to send a query to the service, especially if he doesn't care about the reply, than it takes for the service to process the received query and do some action based on the result. This can then lead to significant mismatch in resource usage in the client-server model which is made more drastic when problematic queries reach the application level and demand complex processing.

3.3 Denial of service

The basic premise of *Denial of Service (DoS)* concept is the prevention of use for some service for legitimate clients due to the service being either hampered or down entirely. This can be done by directing problematic network traffic towards the service. On a deeper level, this whole concept is quite complex as there are various attack methods available that can target different network protocol stack layers and then specific components within those layers. What makes a particular DoS-attack even harder to define is the case that DoS-attack campaign can incorporate various attack methods at different times or even simultaneously to make defending against it harder. There are two major categories for identifying DoS-attacks on a higher level, though: the number of attacking entities and the base method of the attack itself.

The aforementioned number refers to the amount of network entities that will generate malicious data traffic towards the victim. Generally, if there is only one attacking node, the specific attack is considered to be just a "standard" DoS-attack, whereas if there are more attacking network nodes, the attack would actually be a *Distributed Denial-of-Service (DDoS)* attack. Nowadays, most DoS-attacks are DDoS-attacks, as they can leverage far more network resources against the victim, where the attack campaigns could include thousands of hostile nodes. The latter category about methods refers to the attack being either *semantic* or being a *flooding* attack. With the semantic attacks, the goal is usually to target specific network protocol vulnerabilities, where protocols have problems in handling certain malicious data messages or unexpected client behavior. When the service then has to deal with this type of traffic and client procedures, it would use a lot more system resources than normally or it could even crash. The flooding method, on the other hand, refers to overloading the victim's system or network link with a flood of legitimate or semi-legitimate data traffic. The victim's overall system might be able to handle some of this traffic, but simultaneously this would draw away a lot of resources from handling normal users

who may then experience service delays or even service inaccessibility.[31][87]

To elaborate further on computer resources, it is easy to understand that a system running a web service would have finite computing power and memory for handling incoming queries. Usually resolving queries requires data searches and possible data retrieval, both accessing the system memory, and may also require some arithmetic computations if new values for the service database are calculated for example. More queries coming in would then simply mean more spent processor time and more memory usage. In regard to network connections, connected systems have some type of a Network Interface Card (NIC) with specialized hardware that handles incoming and outgoing network traffic and then forwards it to the OS or from it to the network. The important thing to note here is that both the OS and the NIC have data buffers which are essentially data packet queues used to store incoming or outgoing communication traffic that is not immediately handled. Especially with DoS, it is possible that these buffers become full when traffic accumulates to the queue at a faster pace than the system can process data coming out of the queues. The result of this overload is that the system will reject and drop new packets that try to enter the buffer. This ”buffer full” phenomenon is basically the case of a network link being fully saturated, where network-related buffers at some point of the data path start to drop packets due to processing limitations or as the incoming packet flow goes beyond the available link bandwidth.[88][89]

As has been implied earlier in this thesis, there are certain factors with the basic Internet architecture and use principles that make DoS- and DDoS-attacks very feasible. For one, IP is a packet-based, connectionless protocol, where there is generally no accountability for source entities if they spoof their IP packet source addresses. Additionally, network control is distributed all over, and security policies and methods vary a lot in different sections of the Internet, as it is a vast network of different networks. This means that attackers could compromise more insecure parts of the Internet and use those sections to enact DoS- and DDoS-attack campaigns against victims elsewhere. Most DDoS-attacks focus on using hacked computers of unsuspecting, normal network users as attackers instead of taking direct control over network infrastructure, though. Finally, the intelligence which affects data traffic engineering is not necessarily located in the network but at the endpoints, which brings about differences in available resources. The network infrastructure may work in a best-effort manner with also having large bandwidth for data transmissions and this can then be utilized by DoS- and DDoS-attackers, where the network itself wouldn’t have the tools to discriminate against hostile traffic. If large amount of network resources are directed towards some singular network node with DDoS campaign, the target rarely has comparable resources to cope with all incoming traffic.[31][87]

3.3.1 DoS principles and address spoofing

The characterization of certain important aspects of general DoS-attacks, which also apply to DDoS-attacks, is presented in Table 4, as this makes it easier to formulate and discuss defense mechanisms against these attacks. In regard to the semantic DoS-attacks, they are similar in principle to exploits against PKI and authentication mechanisms in a sense that they often rely on victims using outdated and buggy software. Dated example in this context is the ping-of-death attack against OS's network manager by sending a large ICMP ping request to it that would be split over multiple IP packets due to network infrastructure limitations on overall IP packet size. The victim could then have difficulties reassembling these split packets with malformed data on the IP fragment offset field in the header, where loading a problematic, reassembled packet into system's memory could cause buffer overflow. In buffer overflow, faulty data is injected to unexpected locations in memory, where it can overwrite critical instructions, which could then crash the whole system. As network protocol and system logic has improved and as serious software bugs have been ironed out, the far more common threat nowadays comes from the flooding attacks.[31][90]

In contrast to semantic attacks, the basis of flooding attacks is the use of legitimate data traffic. Perhaps the simplest case is just the attacker leveraging traffic generator resources that would exceed the perceived service capacity of the victim's network link and then fill it by sending bogus UDP messages for example. More advanced flooding attack schemes exploit certain protocol behavior or procedures in the network level when dealing with specific legitimate protocol messages or queries. Flooding attacks can also target application level entities such as HTTP servers. For example, HTTP service could suffer from congestion if attackers would send in abnormal amounts of specific queries, where each query would require a taxing database search in order to produce the requested Web-page. These advanced flooding attacks are discussed in more detail later in this section.[31][87][91]

Another important characteristic from Table 4 is the source address usage. The most straightforward way for the attacker would be to use the actual IP addresses of the attacking network nodes but this has several major drawbacks. For one, if the attack traffic comes from limited amount of source addresses, this traffic is easy to pinpoint at various levels on the network in order to *filter* it out. The concept of filtering is discussed in depth later in this section, but basically it is an essential DoS defense mechanism, where a network node in the attack path can reject problematic traffic going through, based on some criteria such as traffic IP source address. The second major drawback with the use of real IP addresses is the accountability, where victim can trace the attackers' resources on their IP source addresses after the attack for legal ramifications and compensation for suffered damages. Therefore, spoofed IP addresses are often used to mask the identity of the attacker, as the IP routing architecture will work with forged source addresses if replies to data traffic can be ignored.[31][87]

Table 4: General DoS-attack characteristics

Characteristics	**Different methods and qualities within a characteristic**	**Further method and quality separation if necessary**
Attacker amount	1 - One attacking node (DoS) 2 - Many attacking nodes (DDoS)	
Basic attacking method	1 - Semantic: target specific protocol and system vulnerabilities 2 - Flooding: overwhelm victim's system with legitimate traffic	
Source address use	1 - Spoofed address 2 - Valid, routable address	1a - Spoof by using randomized, valid source addresses from the whole IP range 1b - Use specific spoofed addresses such as addresses which lead to nowhere or are the same as destination address
Attack rate	1 - Constant 2 - Variable	2a - Increasing 2b - Fluctuating such as bursts of attack traffic periodically
Attack traffic characterization	1 - Can be characterized by used protocol or port for example 2 - Cannot be easily characterized	
Attacker persistence	1 - Persistent 2 - One-off	
Victim Type	1 - Specific application 2 - Victim's system 3 - Victim's network 4 - Network Infrastructure	
Attack impact	1 - Degradation of service 2 - Service shut-down	

Normally networked systems will add valid IP source addresses to outgoing data traffic automatically based on the real IP address of the sending network interface, but a packet manipulation software such as hping can be used to alter outgoing packets before they are forwarded deeper into the network [92]. Additional way to alter address or port information on the transport layer and above is the use of raw sockets, where arbitrary protocol scheme can be utilized on top of IP [93]. Utilizing packet manipulation tools or raw sockets usually requires root access, however, which limits the amount of computers attacker can use with source address spoofing. The basis of how to choose these addresses upon transmitting the packet varies: the address could be randomized from some selected IP address pool or over the whole IP address space based on some random distribution, or specific, confusing or problematic source addresses could be used such as address ”0.0.0.0”. The problem with utilizing a selection of special, spoofed IP source addresses is that they are equally easy to notice and filter than if the attacker would be using a small selection of real IP addresses as sources. This usually leads to attacks with spoofed sources randomly selected from a very large pool of IP addresses spread all over the world.[31][87]

Finally, as the most nefarious method of masking his own IP address, the attacker could utilize willing, or far more often, unwilling but oblivious proxy entities which either generate DoS-traffic based on the attackers' commands by themselves or forward the attack traffic from the attacker towards the victim. With this type of scheme, the proxies could be made to utilize spoofed source addresses similarly to the case where attacker himself would do it to his own source address to further complicate defending against the attack. This process relates to the creation and utilization of *botnets* with DDoS principles, and to *reflection-* and *amplification-attacks*, all of which are discussed further in the following subsections.[31][87]

In addition to noting the source selection, Table 4 presents characterization based on general attacker behavior and on additional features of the attack data traffic. When the rate of attack traffic is considered, the simplest approach is to send as much traffic as the source is capable of transmitting, in a high and stable rate, towards the victim, where the short-term disrupting effect can be drastic, but the attack can then be noticed and reacted to quickly. The alternative to this is to vary the attack traffic rate and try to mimic behavior of legitimate users, where especially with DDoS, many seemingly innocent clients are draining the victim's resources, as there are just so many of them. This also connects to the overall attack duration, as very high traffic rates may be available for the attacker only for a short duration. For example, if compromised proxies are used as middle-men without their knowledge, even they may notice if their network resources are used to the maximum and then disconnect from the attack campaign. For comparison, widely distributed, low rate DDoS can be persistent more effectively, as it is more difficult for the victim and unwilling proxies to pinpoint the attack traffic.[31][87]

When incoming network flows of the attack are examined more thoroughly, it is wise to assume that there is usually some higher-level protocol or protocols that are

attached to the flow, as there is no point in sending just empty IP packets, especially as they can be filtered out quite easily. Flow features such as used high-level protocols, their payloads and the source and destination ports can be checked to filter out unwanted traffic. As is the case with data rates, though, clever attackers can try to utilize same ports and protocols as normal users. This is quite problematic, as most online services offer a few specific services on the standard ports such as hosting a DNS server or hosting web sites with a HTTP server using the well-known ports of 53 and 80 respectively. It is also good to remember that the network nodes on the path won't access this higher level information at least by default, so the attack traffic may go unnoticed until it is very close to or at the victim.[31][87]

From the victim's perspective, Table 4 shows two distinct characteristics which are the specific attack target component in the victim's system (or connected to him in some way) and the effect to this target. In a narrower DoS-attack, only specific applications on the victim's system may be targeted, but as was mentioned before, many web services have some singular purpose, where affecting this main process will then affect the whole system. With more generalized DoS-attacks, the target is the whole host system of the service, where the goal is to deplete the network resources or the computing resources of the victim. This would lead to any hosted application or service being congested or even inaccessible. With cloud hosting, it is also possible that attacking a singular host in the cloud could affect other services there as the services could be hosted on the same server, behind the same network link. If the scope grows larger, attacks could also target critical network elements near the service such as nearby routers that direct traffic to the victim's network segment, or with massive scale attacks, more distant network infrastructure such as BGP routers. Central Internet infrastructure is well provisioned, though, so serious negative effects on these network parts demands substantial resources from the attackers, which could be possible if they are state sponsored for example. As for the result of a particular DoS-attack, most of the time with flooding attacks, the service may not necessarily crash, but major degradation of service quality is often achieved, which in practise could feel like the server is down from legitimate client's point of view.[31][87]

If the motivations behind DoS-attacks are examined, financial gains are an important factor. Criminal organizations or money-hungry hackers could extort money from web service hosts who fear impending DoS-attacks or wish for a ongoing DoS-attack campaign to stop. It is also possible that a competitor for some web service would buy DoS-attacks as a service against this victim in order to drive them out of business. Additionally, spite or a need for revenge may cause some individuals to enact DoS-attacks against former spouse or employer for example, but with these cases, the attackers usually have limited resources at their disposal. Political motivations may also lead to people such as terrorist groups of hacktivists to employ DoS-attacks. Finally, most dangerous groups within DoS-attackers are likely state-backed specialist groups who have plentiful resources and use DoS-attacks for political leverage and as a part of cyberattacks or hybrid warfare doctrine. It is also worth noting that not all DoS-attacks are intentional. It is possible that sudden flash mobs could

overwhelm ill-prepared web services if the service attracts traffic based on fresh referrals from some popular website or from news. More benign example of this is likely the case where commercial web service holds a sale event and then abrupt surge of interested customers comes in causing congestion or even a shut down, as the server is overwhelmed.[87]

3.3.2 DDoS principles

With the general server-client model on the Internet, servers are usually focused on doing a specific task and are provisioned to handle reasonable amount of simultaneous clients from the get-go. The limiting of non-essential functionality on these servers makes this easier compared to standard client computers of an average Internet user receiving queries. This can make it difficult to overwhelm the service resources with a very limited number of hostile clients. Because of this, attack campaigns will often utilize hundreds or even thousands of simultaneous attackers in order to really cripple targets. Recruiting large number of these attackers, so that they would act willingly and without compensation, is likely impossible unless there are wide-spread political motivations against the victim. This has led to many DDoS-attack campaigns to utilize *botnets*, where the attacker exploits security weaknesses of a large group of regular network nodes called *bots* (or hosts) and gains at least limited control over them. These captured nodes can then be used to forward or generate attack traffic towards the DDoS-victim based on the commands from the botnet controller. Another big benefit of using botnets is the fact that they mask the attackers IP addresses, especially if address spoofing is mixed in, as the victim sees the bots being the attack source. It is also worth noting the more tech-savvy criminals could rent their botnets for attackers who have less technical experience. To illustrate, the basic structure of a botnet is shown in Figure 16.[31][87]

In botnet DDoS-attacks, the process starts with the attacker compromising a group of unaware bots who are often outdated or simple network nodes that run older software that has security faults or don't run proper security mechanisms at all, while still being connected to the Internet. The attacker may find and recruit these machines for example by port scanning, from vulnerable node-lists spread among cyber-criminals and from phishing campaigns. In port scanning, attacker would usually send either TCP-SYN-messages or UDP packets to a range of ports on the target node, which would either respond with proceeding with the 3-way TCP handshake procedure or by sending ICMP "port unreachable" error messages if the UDP port is not available or with nothing if the opposite is true [94]. This would then enable the attacker to note open ports and also note deprecated or problematic services by which a worm-type malware could be injected to enable unauthorized access and further network infection. It is possible that scanning for potential bots is done manually, but usually this process is automated and also utilizes bots that have been captured earlier to participate in the scanning process to widen the search for new bot candidates. With phishing, the attacker could send spam emails to wide

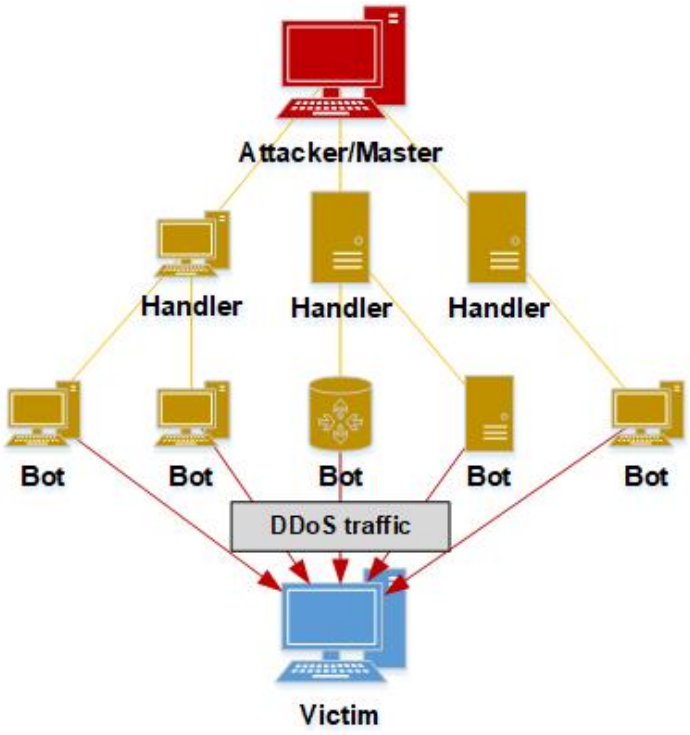

Figure 16: DDoS botnet structure

array of network clients with an attached link which runs an exploitative trojan horse-type program masquerading to be something benign, but would in actuality compromise the system if the link was opened [46, pp. 433–448]. In any case, the methods can be similar as with gaining unauthorized access in PKI, where attacks often rely on human error, on system misconfiguration or on people using older, unsafe protocols and software.[31][87]

After the attacker has gained access to a large number of bots, he must devise a control method to initiate the actual DDoS-traffic. It is possible that the injected code on the bots is set to automatically start attack traffic at certain times or based on some system event, but often attacker requires more hands-on control over the generated traffic, so he may utilize *handler* entities on the bot network. Basically, handlers act as a proxy between the master and the bots to conceal master's identity, where the handler usually is some common web service that can be manipulated by the master. Two widely used examples of handlers are Internet Relay Chat (IRC) servers and HTTP servers. IRC is a web service enabling textual communication via specific chat channels on the IRC server, and when these servers are used as controller scheme middle men, the program exploit on the bot will connect it automatically to some IRC server and a specific, hidden channel there, where the attacker will communicate with coded messages to control the bot's behavior. As IRC is a legitimate service and the control information exchange rarely takes much bandwidth, this communication may be difficult to spot. With HTTP servers as handlers, the attacker maintains a

web page with embedded code that contains the attack and bot control commands, where the bot is programmed to retrieve this web page periodically. Similarly to the case with IRC, using HTTP is a common occurrence for an arbitrary network node, so spotting illegitimate uses of it could be problematic. It is also feasible to use Peer-to-Peer (P2P) methods to communicate the bot control data, where there is no central control and where the bots maintain databases of contact points from where control data and malware updates can be retrieved.[31][87]

The level of access the attacker has on each bot affects how demanding tasks the bot could make in the DDoS-campaign. Most shared computer systems enact the principle of limiting user's capabilities to only what is necessary, which means that if user credentials or access to some large network node becomes compromised, the attacker may not have gotten root access but rather the privileges of a standard user, who has limited options for network traffic control. Address spoofing done by the bots will make tracing them much harder, but this generally requires that the attacker has root access, as spoofing often requires software that does low-level and abnormal packet manipulation. The non-root users can usually open network connections with common protocols from most legitimate ports, though, which makes them perfectly adequate DDoS-traffic sources if spoofing is not necessary. Lastly, if the attacker has very limited, guest-type access, he may only be able to create very specific network traffic such as utilizing TCP connections for HTTP traffic, but this can still be dangerous. On this level, attackers could manipulate web page scripts within the browser sandbox for example to create excessive traffic. No matter how the botnet was recruited or how much access the attacker has to bots, flooding random traffic towards the victim's network at full bandwidth may be effective for a short while, but this often makes the traffic easier to filter out and it could also be noticed quickly by the bot's legitimate controller. For these reasons, masking the traffic to be similar to standard network traffic and exploiting protocol performance weaknesses at the same time can make the bots both harder to detect and more effective in a sense that performance loss to victim per sent DDoS-traffic volume is improved. Going beyond just basic traffic flooding schemes will lead to the discussion about advanced flooding attacks in the following subsection.[31][91]

Finally, to demonstrate real life cases, example botnets from this decade that were successfully used to cause harm include Chameleon and Mirai. Chameleon was found on 2013 and it was though to encompass around 120000 bots. The infected computers would regularly visit sites that had click- or page visitation-based advertising without the bot owner's consent, where each click or visit would generate income to the site from the advertiser. The Chameleon bots used Windows systems of regular Internet users, where the malware could run JavaScript and Adobe Flash (coding- and scripting-languages for web pages), which would then enable hidden connections to aforementioned websites, where each connection would masquerade as a legitimate site visitor. Although this case was not connected to a DDoS-campaign, it works to illustrate creative use of botnets, where the final victims were actually the advertisers.[95] In the more standard DDoS-context, the example is Mirai. It

was discovered in 2016 and was estimated to have between 360000 to 10 million bots, where the network was composed of simple IoT-devices that had not changed their default credentials upon installation or just had poorly selected access credentials. Mirai used port scanning methods to find these devices and then used a brute-force approach to test a list of credentials against the device authentication, and upon successful access, would inject malware to the device which would make it participate in the DDoS-campaign.[8][96]

3.3.3 Advanced flooding attacks

Network protocols that are not strictly connected to security procedures generally base their functionality on commonly expected client or peer behavior, where abnormalities in this behavior are attributed to network congestion, packet loss or other factors that are not caused by the client's or peer's hostile intent. This unfortunately forms the basis for advanced flooding DDoS-attacks, where protocol behavior that was originally meant for congestion control or packet reassembly for example could be exploited with tailored DDoS-messaging to drain victim's system resources. These types of attacks are possible both on the network level and on the application level. On the former, the attacks, at least without redirection schemes, are usually based on exploiting certain types of TCP messages. On the latter, the attack method often relies on demanding or abnormal HTTP- and DNS-queries. [31][87]

With TCP-based advance flooding, perhaps the most common case is the *SYN-flood*. As was shown in Figure 6 in the previous chapter, the TCP connection is initialized by a 3-way handshake, where the first step is for the client to contact the server by a TCP message with the SYN flag on. The server would then respond with a affirming TCP message with the SYN and ACK flags on, and at the same point reserving system resources for the to-be-formed, full TCP connection with the connection then being "half-open". In SYN-flood attacks, the attacker doesn't react to the SYN-ACK responses or spoofs the SYN source which leads to the victim not receiving the final ACK messages which would finalize the TCP handshake. This would then drain the victim's system resources as he has to maintain many half-open TCP connections, where each connection awaits for an ACK message that never comes.[97] Additionally, messages with SYN-ACK, ACK, Reset (RST) or Finish (FIN) flags that don't belong to ongoing connections could be flooded towards the service, where the service would then need to use resources to decipher what to do with these messages. To clarify, RST and FIN messages are used to either request TCP connection reset or tear down respectively, and they are part of normal TCP connection procedure. Attackers can also create fake TCP sessions by completing the initial handshake in a proper manner and follow it with problematic messaging that would request lot of packet re-transmissions or exploit the TCP congestion window. An example of this would be optimistically sending ACK packets to the server even if the client has not yet actually received the corresponding packet that is to be acknowledged. This would then lead to the server transmitting

data faster and faster which could cause degradation of service to other clients.[98][99]

In the application layer, one straightforward attack strategy is just flooding the service with queries that take a lot of resources to process. With HTTP, these can be standard GET and POST-type queries which are used to retrieve web resources or to update a resource on the service. If these actions target large and complex resources, the load for the service can become quite high [17]. Multiple HTTP requests in a HTTP session could even be embedded to single data packet, so that it would be harder to detect problematic behavior compared to a large number of consecutive, separate GET or POST requests. As HTTP can also be used to upload or enter data to the HTTP server, attacks could target the database that manages the web pages hosted by the particular server. This can be done for example with Structured Query Language (SQL) injection, if the database control and maintenance is done with SQL. Using this method, the attacker can send in specifically formulated SQL commands as a text input to some benign web form, which then forwards these commands for processing. This injected control data could then slow, crash or even delete the database and would therefore bring down the targeted web service altogether.[100] With DNS, the simple, direct attack path would be flooding in recursive queries, if the targeted DNS server allows this, where the query would then be propagated further in the DNS system, demanding more and more resources from the whole DNS infrastructure.[87]

In contrast to sending in as many queries as possible, attackers can also exploit HTTP with slow responsiveness and delays in multiple simultaneous HTTP sessions, where the goal is overload the system's capability to keep up a large pool of concurrent, problematic connections, in a similar manner to SYN-flood exploiting half-open TCP sessions. Examples of HTTP delay attacks include the slowloris- or slow headers attack, where HTTP-requests will contain commands and parameters line-by-line with the Carriage Return Line Feed (CRLF) notation marking the end point. If a request is missing the final CRLF note to mark the end, the server may end in a waiting state and expects more commands from the request until it hits a timeout. Attacker can then slowly feed request data in accordance of this timeout period to keep the connection going on while it takes up service resources.[101] In another delay-based attack example, the attacker sets the HTTP-request's content field, which denotes the message content size, to some large value. He can then upload this content at a very slow rate while the server has to wait patiently until all of the data has been uploaded, where multiple simultaneous HTTP sessions of this type will start to degrade the service performance.[87]

As another attack method both on the network layer and on the application layer, protocol data can often be fragmented over multiple IP packets which was the case with the semantic ping-of-death. Even if the goal is not to crash the service, reassembling UDP-packets, HTTP-requests, ICMP-messages and TCP SYN-messages that are split over multiple IP packets may cause lot of processing overhead for the receiver, where a large flood of these packets becomes quite problematic. It can be said that

especially in the case with DDoS, the spectrum of potential threats is quite large, as DDoS-attacks can combine various attack vectors and can also utilize large amount of network resources with botnets. What makes things even more complicated is the possibility to use reflection and amplification for the DDoS-traffic.[31][87]

3.3.4 Reflection and amplification attacks

In order to hide the identities of the attackers or specific bots, DoS-attacks may utilize traffic reflection. This usually involves sending in queries to third party services with spoofed IP source address, where this forged address actually directs towards the victim to make an impression that the victim sent the query in the first place. This makes tracing DoS-traffic more difficult, as first the legitimate service who sent the unwanted queries has to be contacted and the trace needs to be continued from there. Additionally, traffic volume increase method, which is often used in conjunction with the reflection, is traffic amplification. Amplifying means that the initiating query or message of the attack traffic reaches some service, which then generates traffic towards the victim that is preferably (from the attacker's point-of-view) much higher in proportion than initial query volume.[31][87][102]

Simple reflection scheme is sending in ICMP echo-requests or DNS queries to legitimate network nodes with the victim's source address. The victim may be able to filter this traffic to some extent as he didn't initiate these queries and requests, but if the reflected response traffic is generated by some DDoS-attack campaign with a lot of participants, it can easily fill the victim's network link. A more complex example is the amplification and reflection DDoS-attack using legitimate DNS servers as proxies. This scheme is illustrated in Figure 17, where the bots create DNS queries to the legitimate DNS servers. The bots spoof the source address for these queries to be the victim's IP address, which makes the DNS servers direct answers towards the victim, as they may not suspect any foul play. DNS response which actually conveys address information is almost always much larger than the respective query, as it contains RRs that have the queried domain name to IP address mapping and likely also RRs about noting the authoritative name server address. This then leads to DDoS-traffic being amplified as each DNS query will result in a larger DNS reply.[87][102]

Amplification attacks can also utilize IP broadcast addresses, and the Smurf-attack is a classic example of this. Network broadcast addresses are used as a destination address when an IP packet is to be sent to every node in the network segment. Each segment should have this address set up and it is generally the highest possible address value in the network such as 1.1.255.255 for the 1.1.0.0/16 network.[103] In the Smurf-attack, attacker would send ICMP packets such as ICMP echo-request to the broadcast address, spoofing the source address for these ICMP packets to be the victim's IP address. This would then direct all receiving nodes in the network segment to direct replies back to the victim. As a concluding note, if one adds reflection and amplification to the mix with advanced flooding attacks, this can

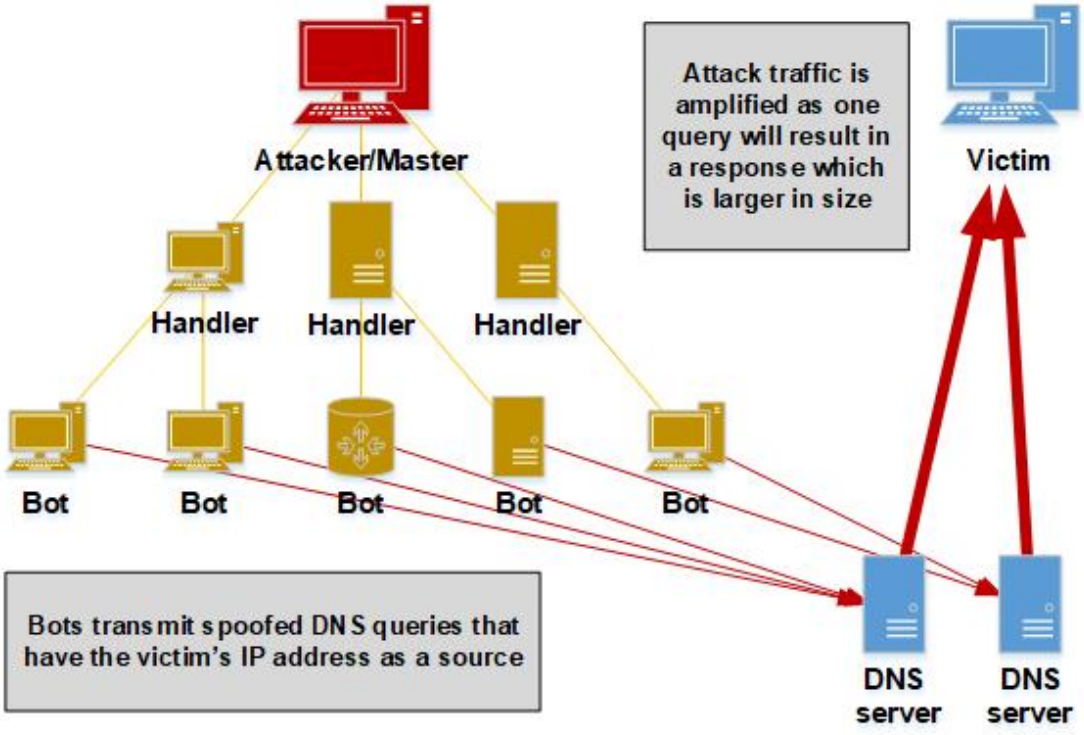

Figure 17: Reflection and amplification DDoS-attack with DNS

create very complex and dangerous DDoS-attacks, where the attack is not easily deterred and the culprits are very difficult to trace.[31][87]

3.3.5 Degradation of service

It is not always the case that DoS- and DDoS-attacks aim to paralyze the victim's system totally or even to a drain resource to a notable degree. Sometimes the goal may be to do a limited resource denial or *degradation of service*, where the overall service capacity is only partly consumed by attackers that don't use much resources at a quick glance, but where the negative effects of the attack traffic accumulates over time. A way to achieve this would be to do Low-rate Distributed Denial of Service Attacks (LDDoS) that try to emulate legitimate user data traffic as closely as possible in contrast to the advanced flooding schemes, where the traffic was in many cases abnormally low or high per connection.[104][105][106]

The motivations to enact DDoS on a very limited scale are likely linked to two aspects. The first is the difficulty to detect these attacks which can make them long-lasting. The second is economic incentive that connects to Economic Denial of Sustainability (EDoS), where attacker aims to make it unfeasible for the victim to keep up the web service due to service payments based on computing resource use, bandwidth costs and customer dissatisfaction about the quality of the service. Competitors

for some service may purchase hard-to-detect, low bandwidth attack traffic from criminal entities, where the traffic simulates traffic from legitimate clients. This will add to the victim's service upkeep costs over time, which may then make him cease operations as profit margins decrease. An example case could be DDoS-attack against some service hosted in a third party cloud, where the cloud system collects fees based on strictly monitored bandwidth use.[105][106]

In regard to emerging cloud services, while they offer an easy way to harness large amount of computing resources on a close proximity to clients for various web services, they are certainly susceptible to DoS-attacks themselves. For one, cloud services usually host multiple different web services on a particular data center and an attack against one of these services could then degrade the performance of all hosted web services, depending on how the computing resource allocation is configured on the host system. In addition, cloud resources could be used as a part of a DDoS-campaign, where nodes in the cloud would generate DDoS-attack traffic. Of course, there are also some positive sides with the cloud concept in the security context, such as using cloud services as an additional system to detect and filter DDoS-traffic.[105][106]

3.3.6 Permanent DoS

In many cases, DoS- and DDoS-attacks will, at most, crash the victim's system, where a simple reboot would bring the service back up, assuming that the attack traffic has ceased in the meantime. Generally, the attacking entity doesn't have authorizing credentials to the victim's system or network and has to rely on network traffic generated outside to disrupt the target. If the attacker has remote access to the victim's system, however, things become far more problematic, as he can enact Permanent Denial of Service attacks (PDoS), where the target system is adjusted or damaged to a degree that major software re-installations or hardware replacement are needed to salvage the service.[107]

The worst case scenario would be an attacker with remote root access, where he could delete and overwrite the contents of the system's hard disk that has the OS data and also contains the software which runs the actual web service. This would then bring the system down until the OS and relevant service software are installed and started again, with also having the downside that possible client data stored on the hard disk would be lost. Even more dangerous attack involves compromising more simplistic and embedded devices such as smaller network routers, where the firmware of the devices is replaced with a corrupted or specifically tailored version by the attacker. Firmware is the low-level software Application Programming Interface (API) that enables OS to access and manipulate the physical hardware, and if it doesn't operate properly, the hardware itself could break or, at best, the specific component just ceases to function. This may require even more extensive repairs than re-installation of software and could mean that the device needs to be replaced altogether.[107]

3.4 Defenses against DoS- and DDoS-attacks

As DoS- and DDoS-attacks can seriously disrupt and bring down legitimate network services, various ways to combat these threats have been devised. On a timing basis, there is research done on what to do proactively before the attack even starts, what to do in a reactive manner during the attack to detect and block it, and what to do after the attack to learn from it to help mitigate future attacks and to find out the culprits to bring them to justice. It is also important to note where exactly the DoS- and DDoS-defenses should be located which is illustrated in Figure 18.[31][87]

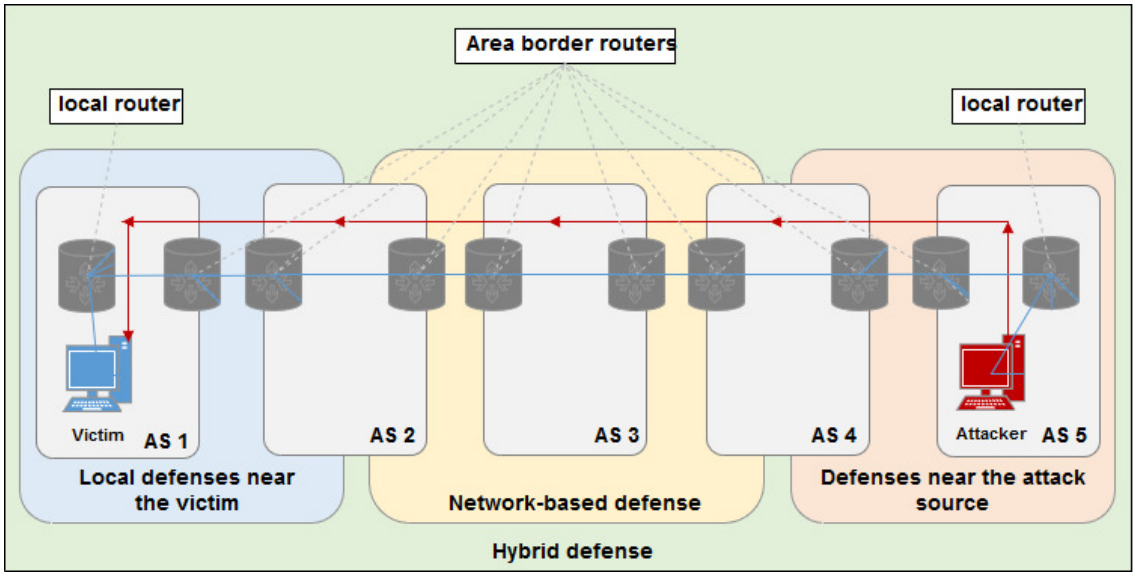

Figure 18: Dos- and DDoS-defense characterization by location

Generally detection and defense processes near the victim are easier, as DoS- and DDoS-attacks are funnel-shaped, where most of the attack data traffic will eventually go through the victim's immediate network. On the other hand, detection apparatus and defenses could be mounted near the attack source to prevent DoS- and DDoS-traffic going to wider Internet altogether, but in these cases, detection may be harder, as there is only limited amount of traffic going out per attacker (or bot) with DDoS. Defense and detection could also be done somewhere in the wider network infrastructure, beyond the immediate networks of the attacker and the victim, but as has been mentioned earlier, entities on this level may have little incentive, regulatory restrictions or limited capability that prevents them doing anything meaningful to the ongoing attack traffic. Finally, the defense processes could be set up simultaneously to multiple of these locations, which is called a hybrid defense scheme. As for what DoS- and DDoS defense and detection means in practice, the attack traffic is often identified by the receiving system monitoring incoming traffic rates, and when they go beyond certain threshold, traffic filtering and rate limiting could be done. These

measures are elaborated further in the following subsection.[31][87]

3.4.1 Traffic filtering and rate limiting

Perhaps the most effective way in practice to stop ongoing DoS- and DDoS-attacks is to *filter* out or limit the problematic data traffic, so that the targeted system isn't stressed to the limit anymore. This usually means that *firewalls* are utilized at some point on the attack path, where the firewall software on a network node monitors the ongoing traffic and then drops packets that are thought to be part of the Dos- or DDoS-attacks. Most firewalls are limited to work with Internet and transport layer information, so parameters for filtering include mainly source- and destination IP addresses, source- and destination ports and some basic data gathered from IP-, UDP-, and TCP-headers such as TCP flag information or IP time-to-live values. This is often enough to deter at least more simplistic DoS- and DDoS-attacks such as flooding towards ports that are not served at the victim's end or large volume of attack traffic from a small set of specific IP addresses.[46][87]

In more complex attack scenarios, problematic traffic that closely mimics legitimate client behavior is far harder to filter, especially if it is part of a DDoS-attack. This can be compounded by the victim only hosting one or a few actual services on his system such as DNS- or HTTP-server, which then receives all incoming traffic to a single port. This limits the possibility to detect and deter attack traffic based on some of the lower layer parameters. In these cases, filtering could go to application level, where the firewall software may have capability to parse some application-specific parameters such as DNS message flags and HTTP request types. This high-level data offers more fine-grained options to handle problematic traffic but processing packets at this level requires more computational resources for the firewall.[46][87]

Also in regard to the case where attack traffic is difficult to separate from normal traffic, aggressive filtering could block legitimate users as well as attackers. This is usually where *rate limiting* comes in, where firewalls impose limits on how many packets of certain type are permitted per time frame. It is possible that most rate limited traffic capacity is taken over by the attack traffic, but at least there is a chance for actual clients to reach the service compared to the case where service ports are shut down altogether. It is often wise to utilize both strict filtering and rate limiting, where service filters out all incoming random traffic and then has some overall limit on the actual, legitimate traffic to the proper service port, where activity below or at this threshold is manageable and does not crash the service. Additional defense mechanisms to separate legitimate clients from unwanted clients is the use of Access Control Lists (ACL), where a client needs to authenticate himself before access to or through the service is granted, as his noted to belong to the list. This may not be feasible for all web services and causes processing overhead, but ACLs can at least help stopping application-level DoS and DDoS, as the attacker cannot access this level at all without authorization.[87][91]

To elaborate on how filtering and rate limiting are set up in actuality, the firewalls that perform them are differentiated first by their location. Firewalls can be deployed to the network infrastructure for efficiency and scalability reasons or they could be deployed to each host specifically for more fine-grained control, in which case they usually are low overhead-software that works closely with the host system's OS kernel or core. Naturally, network-based and host-based firewalls can be used simultaneously. In both cases, the firewall has streamlined access to handle incoming and outgoing network data packets as quickly and effectively as possible before they are processed further. The actual filtering and limitation decisions are based on rules placed on the firewall software, where each rule can denote what action is to be taken for each incoming or outgoing data packet if it has certain characteristic such as specific packet destination and source addresses or ports. More simplistic firewalls are stateless, which means that all rules are generally set beforehand, whereas more complex firewalls can be stateful, where the firewall can track the state of different connections going through it. Additionally, with stateful firewalls, rules could be changed on the fly and could even be triggered to come into effect or be disabled automatically based on some system events or detected traffic characteristics. It is also good to note a couple of key terms with filtering: *egress filtering* and *ingress filtering*, where with the first, firewall stops traffic from leaving local network to outside based on some criteria, and where with the second. firewall drops unwanted traffic coming from the outside to the local network. Very common case with egress filtering is the firewall dropping outgoing packets that do not have IP source addresses that belong to the local network, which is a great help in fight against IP source address spoofing.[46]

One additional, major decision with filtering and rate limiting is where the firewalls are actually located on the Internet infrastructure, where the defense location selection was shown in Figure 18. If they happen to be near the attacker, the traffic could be filtered out by the destination addresses which connects to the victim, but as was noted, the traffic volume per attacker may be quite low if they are dispersed over many different networks. Usually, however, the relevant firewalls are on victim's immediate vicinity, where source addresses could be used for the attacker identification but where the firewall often bears the full force of the attack traffic. Firewalls could also be somewhere on the larger Internet, but the network nodes there may have very limited capability to limit and drop traffic, as firewall rule adjustments at that level can easily affect innocent network users. As a final note, it is good to understand that filtering on a local level doesn't prevent the system's network link for becoming fully saturated by attack traffic, where packets are then dropped at the other end. Defense against this kind of attack basically means that filtering should be propagated upstream. Perhaps the most relevant hurdles to enact effective filtering beyond the victim's own system and his network are the distribution of system control on the Internet and the communication issues between the victim and upstream entities. The process to note management of some distant network about the DoS- and DDoS-traffic they generate may be cumbersome, and it may take long time before any preventative or blocking measures are taken, if they are taken at all.[87][91]

3.4.2 Basic DoS defense preparation and attack detection

A few basic proactive measures will be necessary to maintain web system security and especially availability. For a start, systems should obviously enable and maintain firewalls so that non-critical traffic and especially unsupported protocols are easily filtered out. It is also good to set sensible network rate limit from the very beginning to avoid total system crashes due to abnormal amount of data traffic. Additionally, the system's security suite and authentication methods should be up-to-date to avoid unauthorized access which could enable permanent system shutdown or other compromising behavior. Although the easiest place to implement security changes is the to-be victim system itself, it is possible to set up fail-safe systems and load balancing if additional server resources are installed. Hybrid defense schemes should also be implemented if possible, at least to an extent that the local ISP and could be contacted in the case of a DoS- or DDoS-attack, where the ISP could do upstream filtering on a larger scale for the problematic traffic. More detailed discussion on structural changes on the network infrastructure and what hybrid defenses could entail follow a bit later in this section.[31][87]

During the DoS- and DDoS-attacks, it is important to actually notice that something is wrong and that the abnormal traffic does not come from regular users, which then gives the victim a reason to enact additional defense measures on top of proactive filtering rule setup for example. One could install advanced intrusion detection systems to look for masqueraded attack traffic and set some DoS-threshold for the network link. The most straightforward attack indicator and detection starting point would then be the incoming traffic level going beyond this normalcy threshold. Reaching this limit could just mean a surge of normal users, though, but it is logical to start thinking about defensive measures at this point, as further traffic will degrade system performance. The previously mentioned stateful firewall could have traffic reaching this threshold as a trigger event, and it could also mean that the system state goes to place, where traffic needs to be examined more closely and the filtering is done accordingly. In addition, it is wise to gather information about previous DoS- and DDoS-attacks, where they might have specific, recognizable features or processes which can be identified quickly.[31][87][108]

If the victim has lots of computing resources in his disposal, it is possible to utilize more advanced attack detection methods that are based on the automation of attack traffic. With DDoS, the participating bots usually have scripted behavior and synchronization scheme which could make their attack traffic distinguishable from traffic generated by human users. In waveform analysis for example, the spectral components of the incoming traffic signal could be monitored for certain abnormal frequencies. If the traffic would have lots of high frequency and medium frequency components, that would indicate an attack, as lots of hostile nodes may try flood in traffic in quick succession, in a scripted fashion. The automated attacks could also

be noticed by relying on statistics about the average user behavior that has been gathered over time, where client behavior per flow is stored as time-series data in the time domain. The system could then monitor the behavior of incoming flows and detect abnormalities based on the expected client behavior near the DoS- or DDoS-attack traffic starting point. A practical example of this would be utilizing a cumulative sum (Cusum) algorithm, where Cusum calculates the deviation between ongoing traffic's local time-series average and the expected time-series average. Flow characterization is usually based on the Internet- and transport layer headers, so it may be necessary to aggregate this information for clusters. This would then make it easier for machine-learning algorithms to detect flow anomalies and do general traffic analysis on a per cluster basis, where the problem dimensions are reduced.[108]

The automated non-human like behavior of the attackers can also be used in the defense on the application level. Attackers could be detected by embedded challenge-response tests, such as identifying some object on a given picture when the web content on HTTP servers is accessed for example. In this case, the bot could not give out legitimate response to the challenge, as complex image recognition is hard to implement with simple and quick algorithms. In addition, web content could include items that are actually not shown on the screen such as objects that are marked to be invisible in the web page code. System would then notice bot behavior as they would interact with this code, as their behavior could be based purely on parsing the web site code instead of acting on what the code represents.[87]

3.4.3 Defenses against SYN flood

A bit of a special case with the DoS- and DDoS-attacks and defenses is the SYN -flood. It is difficult to utilize strict filtering with TCP SYN packets, as harsh limits there would eliminate legitimate TCP connection formation. In many cases, it is also difficult to base user behavior analysis just on a single initiating SYN packet. Simple workarounds involve reducing the time the receiving system waits for the respective ACK but this can only offer limited help, as legitimate clients would also need some time to finish up the TCP handshake due to network delays and congestion. Therefore, two more sophisticated defense mechanisms have been developed to limit the creation of half-opened bogus TCP connections on the server side, which are the use of SYN cookies and the use of SYN proxies.

SYN-cookies base their functionality on the server using specifically crafted TCP message sequence numbers which are generally used to keep track of the TCP message ordering. As the client first send the SYN message, the server will not fully enable the half-open TCP connection for the OS SYN queue, but sends back a SYN-ACK with the initial sequence number based on the following t, m and p values, where they a calculated as follows:

- Parameter t would be a slowly increasing timestamp value

- Parameter m would be maximum segment size of the system SYN queue entry
- Parameter p would be a cryptographic hash from t value and the source and destination ports and source and destination IP addresses which are used with the connection

The actual forwarded sequence number would then be a 32-bit concatenation of a 5-bit value of t times modulo 32, a 3-bit value denoting m and a 24-bit hash p. If the server then receives an ACK message with the properly incremented sequence number based on the crafted, initial sequence number from the server, the server can add the connection to the SYN queue as it is thought to be legitimate. This is basically a challenge-response scheme, where some extra effort is required from the client before resources are fully committed to him. A downside here is that the server will disregard possible TCP options that would be communicated with the initial SYN, which may limit how well each connection would perform.[109][110]

In contrast, the SYN proxy will act as transparent proxy entity between the client and the server to validate the client before the service allocates full resources to the TCP connection. The connection formation in this case is illustrated in Figure 19, where it is assumed that the SYN proxy has quick access to the server it protects. After receiving the initial SYN message, the SYN proxy will note TCP connection parameters and then sends a SYN-ACK response back to the client on behalf of the server. Client will then respond to the SYN proxy masquerading to be the server with an ACK message, if he's legitimate. The SYN proxy can now connect the SYN and ACK messages from the client and notices that the client response is valid. It then proceeds to do the 3-way TCP handshake with the server on behalf of the client, using the data from the original SYN message. After this procedure, the actual TCP data transfer can begin between the client and the server. The upside here is that there are less limitations to the TCP parameters compared to using SYN cookies. It is good to note that SYN proxies can be done either as separate servers in front of the protected server or within the protected server's operating system kernel. If the SYN proxy is totally separate, it could leverage more computing resources, but it then needs to adjust the TCP message sequence numbers to comply with the TCP sequence numbers of the server it protects, as the server chooses these numbers independently. With the SYN proxy residing essentially as a separate network management process in the server, this sequence number adjustment can be more immediate and straightforward, but the proxy will then tap into the server's own computing resources.[111]

3.4.4 Defense by routing configuration and network restructuring

The possibility to adjust network structure to help against DoS- and DDoS-attacks was mentioned earlier in this subsection and what this actually means is changing the network topology to add more physical resources to the local system or to the wider network. The first point here is the adoption of fail-safe mechanisms which

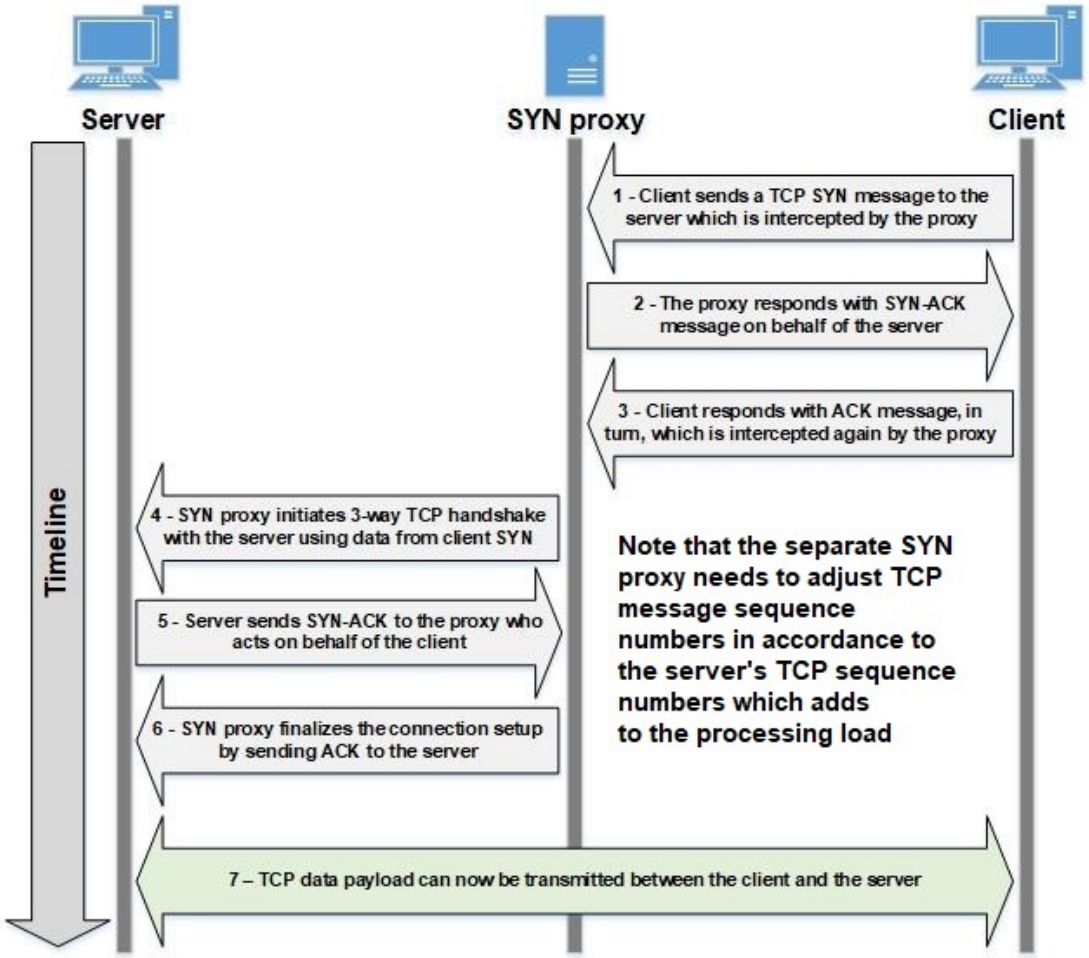

Figure 19: TCP 3-way handshake with SYN proxy

usually means running back-up servers where traffic can be directed if the original server goes down. In addition to the installation of the actual servers, this also connects to routing and DNS, as this means adding optional routes and updating DNS information dynamically in the case of service downtime, if the back-up servers have different IP addresses. Another option is to add more servers, firewalls and routers to the system on a permanent basis to achieve load balancing. Here the incoming traffic would be split over these new servers, so that incoming traffic quantity that did overwhelm 1 server for example would not overwhelm 3 or 5. Depending on how the addressing is handled with these new servers, there may be a need for some load balancing schema for routing that will then split the traffic equally among the servers.[87][91]

Far more complex solution would be to add a traffic absorption overlay system on top of existing network infrastructure. There the service could not be accessed directly, but there would be various levels of proxy machines, where the actual service address and connection would be established only after the client has successfully queried a randomized sequence of these proxy entities. Obviously there could be

large connection establishment delays and deployment issues, but this may be a suitable option for small, critical web services.[112]

For more practical traffic absorption, rerouting mechanisms or cloud services could be used. It is possible to use multihoming, where there are different route possibilities towards the service and the routing data is dynamically updated to lead some or all of the problematic incoming traffic to either black hole addresses that will just receive the traffic and do nothing or through an over-provisioned cloud service which can handle large amounts of traffic. Usually re-routing requires some signaling in the network, as the existence of an attack needs to be communicated to other network nodes in order for the route updates to trigger. As was alluded earlier, this links to the *hybrid* DoS- and DDoS-defenses, where multiple sections of the network infrastructure take part in detecting and dealing with the DoS- and DDoS-attacks.[87][91]

3.4.5 Hybrid defense and upstream traffic filtering

The most effective way to combat DoS- and DDoS-attacks is to combine both proactive and reactive defense methods on many locations on the network infrastructure to form a good enough level of overall security, as each point of defense has its own strengths and weaknesses. The key point here is the communication between defense systems on each location, so that for example the immediate router near the victim or the victim's system itself could send information about the DoS- or DDoS-attack to upstream services, such as the nearest AS outside local network. This information could then be propagated to each AS further on the path towards the attacker as is seen on Figure 18. There are no widely used standards for hybrid DoS- and DDoS-defenses but various commercial actors have set up their own systems, which mostly try to leverage cloud resources. Unfortunately the exact nature of these defense systems is usually a commercial secret, but some basics of the state-of-the-art in this context are available from recent patents.[87][91]

For many modern hybrid defense schemes, cloud resources can be used as reverse proxy servers where the access to the actual service seems to work normally but the communicated data is actually routed through this reverse proxy. These proxy nodes are generally well-provisioned with enough link bandwidth and computing resources to do effective DoS- and DDoS mitigation, helped by information about the DoS- and DDoS-attack characteristics received from elsewhere in the network. Two examples from industry, from Arbor Networks, Inc. and from Cloudflare, Inc. respectively, will illustrate the workings of the hybrid DoS- and DDoS-defense. The basics of the first example, utilizing specialized *DoS scrubbing centers*, are shown in Figure 20.

In the aforementioned system, there are network traffic sensors placed on key locations on the service's immediate vicinity where there are sensors on the local network router and, if it is permitted, sensors on the service's ISP's network routers. If sensors detect abnormal data traffic such as impending DDoS-attack, they can

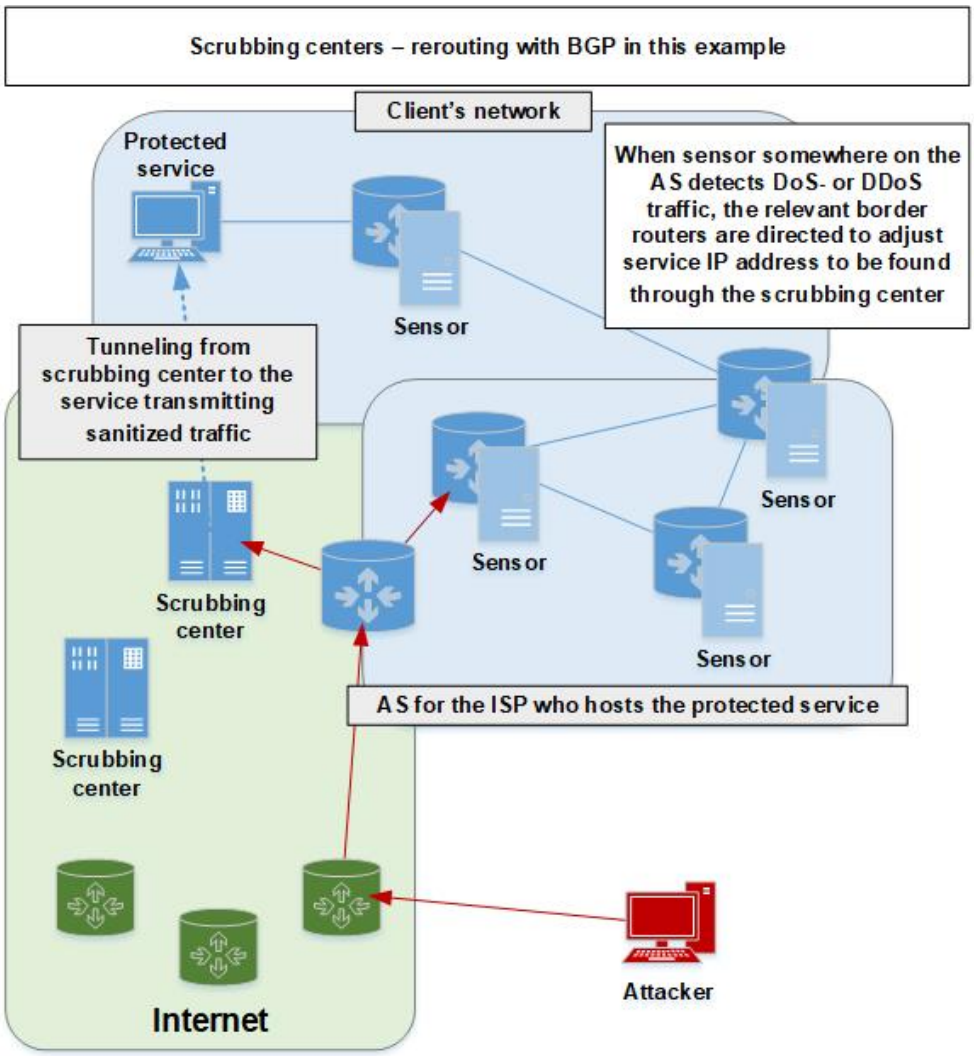

Figure 20: DoS-scrubbing system architecture

signal specific upstream entities for routing adjustments in an on-demand fashion. For example, in the victim's local router, there could be a separate device or software which does port mirroring on the router and can then accurately determine what kind of traffic goes on and if some traffic rate threshold is exceeded.[113]

The sensor devices can be either DNS system components or specific BGP routers which then notify other system nodes about the DoS- or DDoS-attacks via UDP messages. In the case of DNS, the DNS database can be updated dynamically to change the service domain name to direct to the DoS scrubbing center address, at least until the sensors notice that the attack has stopped. Another option is to utilize BGP route injection, where the sensor signaling instructs specific BGP routers to adjust their routing databases so that the IP address of the service is advertised to be reachable via route by which the DoS scrubbing center is located. For example in Figure 20, the ISP's area border routers could withdraw the route advertisement for

the client's local network and simultaneously the AS where the scrubbing center is located would advertise that the path to the service is through it.[113]

The scrubbing centers are servers hosted by a cloud service that should have ample resources to deal with the incoming DoS and DDoS-attacks. As the sensors instruct the routers or DNS system to enable rerouting, they inform the scrubbing center about the attack traffic by sending traffic fingerprints such as problematic IP addresses and port numbers, among other characteristics, which then help the center to filter and rate limit the relevant traffic. The centers could also employ more advanced defense measures such as parsing application level data from the traffic and doing additional filtering based on that. With DNS system adjustments, the scrubbing centers should obtain and utilize the old IP address of the service which is then used as an initial destination for all incoming data traffic towards the protected service. The sanitized traffic is then routed towards the actual service with the scrubbing center using the new adjusted IP address of the protected service instead of the old address. On the other hand, the BGP method requires tunneling between the scrubbing center and the client service, as directing cleaned traffic to the client via normal routing means that traffic will just return back to the scrubbing center. Tunneling protocols such as Generic Routing Encapsulation (GRE) allow the traffic at the scrubbing center to be encapsulated and a point-to-point link to be created between the client and the center over IP, through which scrubbed traffic can then pass.[114] In this latter case, the client would have specific IP address for this tunnel endpoint that the scrubbing center knows, so that the tunnel can be established easily.[113]

The second example hybrid defense scheme is presented in Figure 21. Here the clients from the Internet will always connect to specific reverse proxy servers which pass the traffic to the actual service. The proxies are controlled by a central control server, and both the proxies and the control server can feed data to the DNS system. Similarly to the previous case, the DNS system can be updated dynamically, where the proxy servers may notice abnormalities in the client behavior for example when DDoS-attack raises the ongoing traffic rate to go beyond some normalcy threshold. This would then lead the affected proxy to notify the central control and the DNS system to adjust the DNS resource records, so that the IP address under attack will be a black hole address and the actual service is allocated a new IP address. In the same manner, if some specific IP is under attack without the involvement of the DNS, the IP address can again be changed so that service is allocated a new one, which is taken into account in the DNS system, and the previous IP address is then set to be the black hole address. This requires that the service has access to address pool of decent size and also that the proxy servers are able to adjust the service IP addresses on the fly. Additionally, if multiple domain names point to a single IP address, the system has the option to distribute additional IP addresses to each domain name, so that possible DDoS-traffic directed to all of these domain names is split over multiple IP addresses.[115]

Important feature with the system from Figure 21 is the possibility to distribute

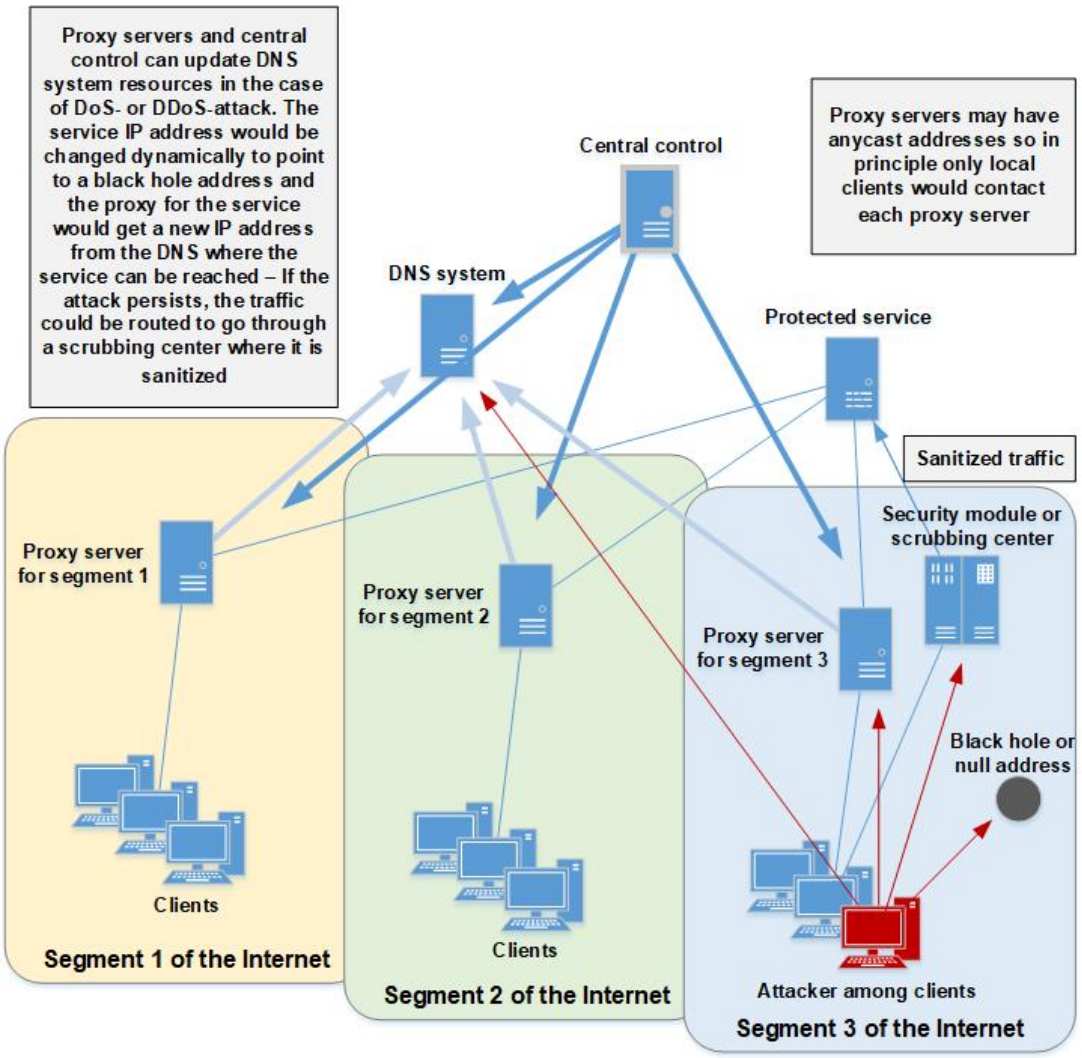

Figure 21: Proxy- and DNS-based hybrid DDoS-defense

proxies with anycast addresses, where the proxies are located on distant geographical areas. With anycast IP addresses, multiple network entities may have the same IP address, where the clients are actually routed by normal Internet routing rules to the nearest node that has the respective destination anycast address.[71] This enables the proxies to maintain information on expected client IP source addresses, where a proxy in a specific region should only get clients from that area, and where sources from elsewhere could be considered spoofed and could be filtered.[115]

The second example system has also scrubbing center-like functionality, where it maintains separate DoS- and DDoS-prevention modules, which could be hosted in a separate location. If the DoS- or DDoS-attacks are persistent, in a sense that the attack traffic will follow the service to the new IP address as the old address is changed to point to nowhere, the system could redirect traffic to the service to go through the security module. These systems have same qualities than the scrubbing

centers in the first example, such as ample network bandwidth, good processing power and link buffers with large capacity so that high traffic volumes are easily processed and there is an option to do filtering and rate-limiting based on the application level data on the network flows.[115]

Finally, the proxies and security modules in the latter example have the option to enact a challenge-response scheme to differentiate bots from real clients. The proxy server could utilize image- or reading comprehension-based tests that were mentioned in the earlier subsection. Example tests could include identifying objects in a picture or deciphering morphed text, both of which are hard to automate with computer algorithms, at least in a rapid fashion. These tests would then be embedded to intermediate web pages that are located at the proxy, where access to the actual service is only granted after a client passes the challenge. Other differentiation methods could include embedding complex computational tasks to the intermediate page, where commonly used web browsers would quickly solve the task but automated scripts designed for simple page retrieval would not.[115]

As a general note, the DNS record adjustments that hybrid defense mechanisms use have lot in common with Content Delivery Network (CDN) basics, and it is not that surprising that many commercial entities that offed DoS- and DDoS-defenses also offer CDN services. With CDN, DNS address resolution is based on the DNS client's IP source address, where the resolved IP address for some service hosting video content for example is chosen to be as close as possible to the client in network topology terms to reduce network delay. Hybrid defense mechanisms can also enact data caching at the proxies which comes from the CDN playbook. Important service content could be cached at proxy servers during DDoS-attacks to enable legitimate clients to access this content quicker.[115][116]

3.4.6 Follow-up measures

From a technical, system-specific perspective, usually, the worst damage that DoS- and DDoS-attacks can cause are certain periods of service downtime where the recovery is possible with just a system restart. This would mean that the aftermath involves mostly trying to learn from the attack in order to identify similar attacks quickly in the future and to track down the attackers by the network traffic traceback. At least this is the case for somewhat detached web services, but the situation would be quite different if services that would control real life infrastructure and transportation would go down even for short duration.

A critical thing for improving the victim's system security is to keep relevant logs about the ongoing data traffic and system behavior, preferably at all times, while the system is running. This will make it easier to go over the experienced attack traffic patterns in detail and will also help observing if some system components were susceptible to high loads and went down unexpectedly. One additional way

to examine past DoS- and DDoS-traffic is the backscatter analysis, where spoofed IP source addresses will lead to the DDoS-victim sending replies to these spoofed addresses, which are often randomized. If this backscatter data is monitored further on the network, it could be used to roughly note attack magnitudes and durations, if the victim's own system didn't have sophisticated logging of traffic enabled for example.[31][87][117]

A very important factor with DDoS- and DoS-attacks is actually identifying the attacker which can then lead to legal actions against him. This is when the attacker's actual IP address should be found out so that the ISP that would usually provide this address would connect the respective address to a person. This is unfortunately anything but easy in most cases. For one, the attacker could have used address spoofing and reflection in conjunction with a botnet, and even if the bots' identities are found out, they could still be far removed from the actual attacker. Taking the infected bots off from the attacker's grasp is a start, though, and this is where actual IP traceback methods could help. The major categories of IP packet tracing are packet marking and link testing. With packet marking, routers on the traffic path add source path information to packets they receive and forward. This could then be used to track the original source, depending from how far the routers started to include source data. With link testing, the routers would go upstream, node-by-node, starting from the victim, where at each step, the router queries upstream routers about the traffic source. This would likely require the routers to keep at least short-lived logs of traffic they have processed recently. Both methods have big implementation challenges as they introduce traffic overhead and it is difficult to get all or even majority of the network operators to update or adjust their systems to support these measures, when there is no clear benefit for them to do it. In conclusion, with more careful attackers, actually reaching the attack originator is often not feasible, but at least it is usually possible to cripple the DDoS-related botnets to make it far more difficult to enact large scale DDoS-attacks.[87]

3.5 The Realm Gateway and Linux network security basics

In order to find out new, applicable security system solutions against DoS- and DDoS-attacks, the Realm Gateway software concept was brought up in the Introduction-chapter. The Realm Gateway is a Python software that runs on the Linux OS and combines the functionality of a DNS resolver, a DNS server and a NAT service, and it can act as a network gateway to a separate web service in some private network, which is supposed to be protected from DoS- and DDoS-attacks while still being accessible from the Internet to legitimate clients. For handling the network security and address translation at the gateway, Realm Gateway relies on the Linux Netfilter framework, which offers access to OS network functions, and is used to bring up and adjust the system's firewall and enact port forwarding and address translation decisions.[15][118]

3.5.1 Linux Netfilter

The foundation of Linux system security is the user authentication and the respective privilege limitation of users by the given authentication credentials. Linux OS maintains the user passwords for the system in a hashed form, which are only accessible to the root user. The OS kernel will then compare given credentials to this list by calculating the hash from the user's input password upon access attempt. As the user gains entry, all programs, files and directories in the OS have access parameters which denote if a user with certain level of access is permitted to read, write, manipulate or run these OS objects. This basically limits what users can do to OS components that they shouldn't have control over. Linux OS can also be accessed remotely if it runs connection services such as SSH server process. SSH for example enables users to connect to the service's port 22 with SSH client, gain access to the system upon giving their credentials and then give commands to the OS via an encrypted SSH connection.[119]

For many systems that are connected to Internet, the main purpose is not to grant remote access to low-level OS items, though, but to offer up high-level application data upon requests such as HTTP pages or DNS answers, or to route data to another node in the network. In these situations, more fine-grained control over incoming data traffic is helpful as these services should be usable by clients without credentials to access the underlying OS. As has been discussed earlier, firewall solutions are used to enforce this control, and in Linux, this can be done with the Netfilter.

The basic functionality of Netfilter relies on intercepting ongoing network events in the Linux kernel and then utilizing specific networking modules which can register callback-functions for these events. Callback-functions can receive other functions and executable code as an input such as networking-related filtering or routing rules which are then applied to the ongoing data traffic. The networking modules with Netfilter are *ip_tables*. *ip6_tables*, *arp_tables* and *ebtables*, where the first two relate to handling IPv4- and IPv6 traffic respectively, and the latter two manage networking on the link layer. The actual data traffic is controlled via rules which are organized into rule chains which are stored in specific rule data tables. Networking modules then access these tables and look up relevant rules which are applied to the traffic in a packet-by-packet manner. The rules in the rule chains can implement various basic traffic shaping procedures such as packet filtering by dropping packets based on IP source address or destination port, incoming packet rate limitations based on similar criteria, packet marking, IP address changing and port forwarding.[118]

The relevant rule tables for most network traffic are *raw*, *mangle*, *nat* and *filter*, and the connected rule chains are *prerouting*, *postrouting*, *input*, *output* and *filter*. Figure 22 shows the simplified IPv4 traffic flow through Netfilter and also when and where the aforementioned rule chains and rule tables are used. If the system doesn't act as a NAT service, the incoming traffic goes first through the raw table and the mangle table before the system makes a routing decision to either direct the packet to higher-level application or to forward it to some other node in the network. In either

case, going through the filter table is an additional step here, where unwanted packets are often dropped so they do not propagate further to the network or propagate to the application level, where processing packets takes more resources. If the traffic goes to a local process, the output path becomes more complicated compared to simple packet forwarding. Basically, additional routing and rerouting decisions need to be made as response traffic may be generated or ongoing traffic may be altered, where the system has to check if the addresses for the new or adjusted traffic are applicable. Finally, the output path offers various steps to do last moment changes or filtering to the traffic before it is sent to the outgoing queue. In regard to NAT, the source- and destination address mapping from port values to addresses and vice versa is done with the nat table on input- and output paths, where Netfilter utilizes system's separate NAT mapping database.[118]

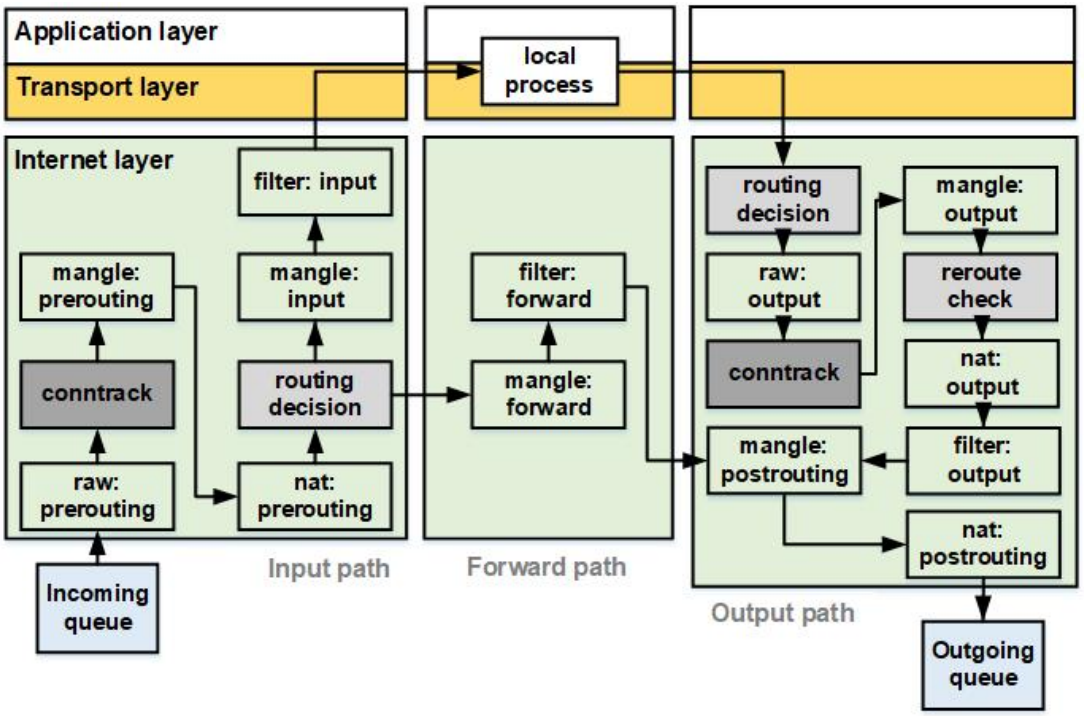

Figure 22: Netfilter IP traffic flow

As an additional feature to track data traffic per communication session such as per TCP connection, Netfilter maintains a connection tracking system. The conntrack steps in Figure 22 show when Netfilter determines and sets the data packets' session statuses, where the communication session is generally identified by the Internet layer protocol in use, the Internet layer source- and destination addresses, and the transport layer protocol- and port information. This tracking scheme can help to maintain NAT functions and also enables stateful firewall properties which can affect filtering and rate limiting for communication sessions. On a base level, the sessions can be marked as "new", "established", "related", "invalid" or "untracked", where the mark

name implies the status of the particular session. A demonstration of the tracking can be the TCP connection setup, where the initial SYN message from a client to a server that runs Netfilter is marked to belong to a ”new” session. The SYN-ACK reply and following ACK reply would then upgrade the session to be ”established”. The following TCP payload traffic would be a part of this session and possible ICMP error messages for this session would be a part of a ”related” session. Netfilter can then use the respective session ID to differentiate between various TCP sessions of the same client, if TCP traffic to certain ports needs to filtered out for example.[118]

3.5.2 The Realm Gateway software

The basic deployment of the Realm Gateway service, when it acts as a gateway to larger public network for some protected web service in a private network is shown in Figure 23. The Realm Gateway itself is a Python 3 software [120] for Linux OS that has been developed for the Aalto 5G project in Aalto University by Jesús Llorente Santos with Hammad Kabir's help for the network security design. The Realm Gateway program code is available for download on the Aalto 5G GitHub page [15], and it has three interconnected modules: DNS forwarder, DNS server and NAT service. The modules can communicate with each other, and with the combined use of the latter two, the main function of the software is to bring up a temporary access point through the NAT to the protected service upon successful DNS query for this service. The DNS forwarder and server modules also utilize *dnspython* library for basic DNS message manipulation [121].[122]

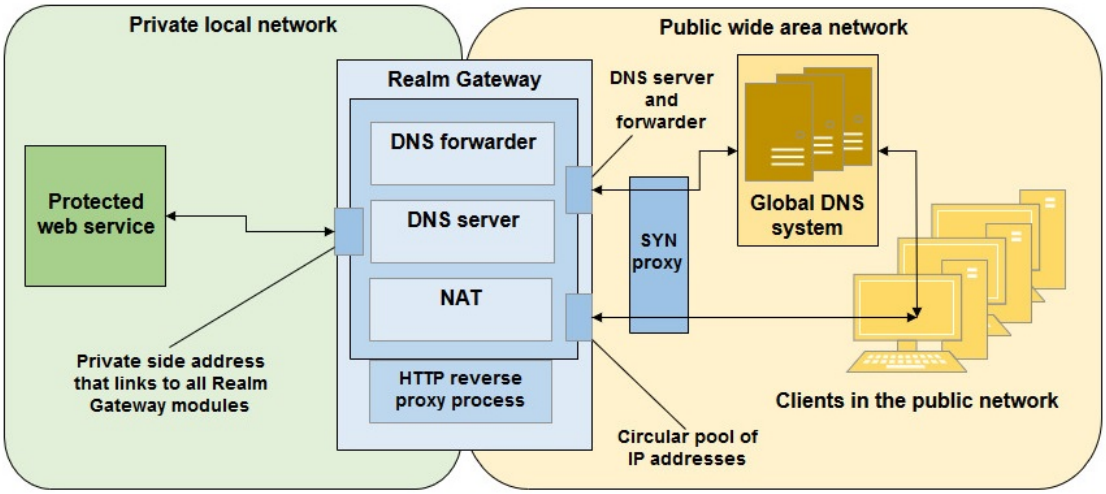

Figure 23: Realm Gateway service protection

To elaborate on each module, the DNS forwarder acts as the first DNS contact point for the protected service in the private network. The protected service can use this module as a local DNS server for recursive queries for public network domain names, if the service needs to access other services on the public network. The DNS

forwarder can query the global or public network DNS systems for the requested domains and can direct replies back to the protected web service, which can then contact the public services through the NAT. In the context of this thesis, this is not thought to be an essential part the Realm Gateway, as generally protected web services would be relatively contained and would not initiate contact to the public network side on their own that much.[15][122]

The DNS server module plays a larger role, as it is the part that actually makes the protected service accessible from outside. The Realm Gateway DNS server works as an authoritative DNS server for the protected web service's domain, where the DNS queries to resolve the protected service's IP address would eventually lead to the Realm Gateway. The DNS server module would then resolve these queries by responding with an IP address from a circular pool of IP addresses available on its public network side. This public IP address is then mapped to be "usable" for connecting to the protected service by the DNS server module in conjunction with the NAT module. The utilization of this address pool would make it harder for DoS- or DDoS-attacks to predict the allocated IP addresses during the DNS resolution. It is also possible to require additional procedures from the DNS client to make sure that his intentions are valid, and that IP address allocation is not done in vain. The DNS server module may require the client to repeat the DNS query with TCP, where a successful connection establishment in this way would ensure that the query source address is not spoofed. Additionally, the DNS server can send a CNAME DNS reply with a randomized, long string being a part of the CNAME response, where a new query with the given CNAME domain would resolve the actual service address via the allocation process. This is essentially a challenge-response procedure, where the Realm Gateway expects the client to respond with a new query based on the unique, client-specific CNAME response.[15][122]

As the client proceeds to contact the protected web service, the NAT module will monitor this process, where the incoming payload traffic from client is sent to the application level to be processed by the Realm Gateway code, where it is logically linked to the client's successful DNS query and marked to be legitimate. This marked traffic is then passed to the Netfilter, where it is allowed to go through the rest of the system and where the nat table rulings will handle the address and port mapping between the private and public address spaces. The Realm Gateway setup may include enacting Netfilter traffic rate limits for incoming DNS traffic, where the NAT and DNS server module could be affected and where the goal is to prevent the system from being overwhelmed on an application level.[15][122]

In Figure 23, an additional important component against DoS- and DDoS-traffic with the Realm Gateway is the separate SYN proxy node. The SYN proxy works, as was discussed in the previous section, as a proxy server to ascertain that incoming TCP connections are legitimate, where the Realm Gateway node itself doesn't have to maintain half-opened TCP connections. This will prevent spoofed TCP traffic from reaching the DNS server module where TCP is used during the client

validation. SYN proxy will also prevent attacker flooding the protected service with TCP messages if he did manage to allocate IP address from the Realm Gateway's circular pool legitimately and aims to use this address for the attack with spoofed traffic. The downside of relying on SYN proxy to this degree is that it is not feasible to provide secure UDP services behind the Realm Gateway. Attackers could send spoofed UDP traffic towards the service if they have been allocated addresses to this service already from the DNS process, where SYN proxy cannot handle UDP traffic. Large majority of security-minded web services are based on TCP, though, as it offers built-in mechanisms to maintain more reliable connections, so the issues with UDP would not be that big of a problem with practical deployment of the system. Providing TCP services behind the Realm Gateway would then mean just blocking the UDP traffic altogether for the circular pool gateway IP addresses.[15]

It is also good to note that the Realm Gateway can utilize a separate HTTP reverse proxy service for handling HTTP traffic, as can be seen in 23. The motivation for the reverse proxy relates to modern web pages having content such as advertising or embedded video which prompts additional DNS queries and network connections when this content is accessed or used. This would create additional stress for the Realm Gateway DNS module, if these additional connections would utilize address allocations from the Realm Gateway's circular address pool. Therefore, the Realm Gateway setup process can enable Netfilter to mark and direct all HTTP protocol traffic to go through the HTTP reverse proxy towards the protected service, bypassing the circular pool address allocation in order to preserve system resources.[15][122]

In more advanced Realm Gateway deployment, load balancing is possible, as can be seen in the setup in Figure 24. The goal of load balancing is to distribute DNS query traffic and access allocation in regard to the protected service to multiple Realm Gateway instances. In this case, there is a *DNS relay* component between the last upstream DNS server and the Realm Gateway DNS server that is marked as authoritative DNS server for the protected service and that will randomize the queries between the multiple first-line Realm Gateways. The randomization process makes it more difficult for the DoS- and DDoS-attacks to target specific Gateway based on the DNS query process results. As the DNS relay or *custom DNS module* is a more lightweight service than Realm Gateway itself, it is able to handle DNS query rates beyond the combined handling rates of a small set of separate Realm Gateways. The workings of the custom DNS module are presented in more detail later in this section.[123]

The Figure 24 also shows two important systems behind the front-end Realm Gateways. The first is a router doing source-based routing, which enables load balancing in the multiple network gateways-scenario. It is usually the case with services behind a NAT that they would have only a single gateway address (the NAT's private side address), where the traffic directed beyond the private network is sent. With multiple NATs acting as gateways, the problem is that with normal routing, the return traffic from protected service to public network could be directed to the wrong

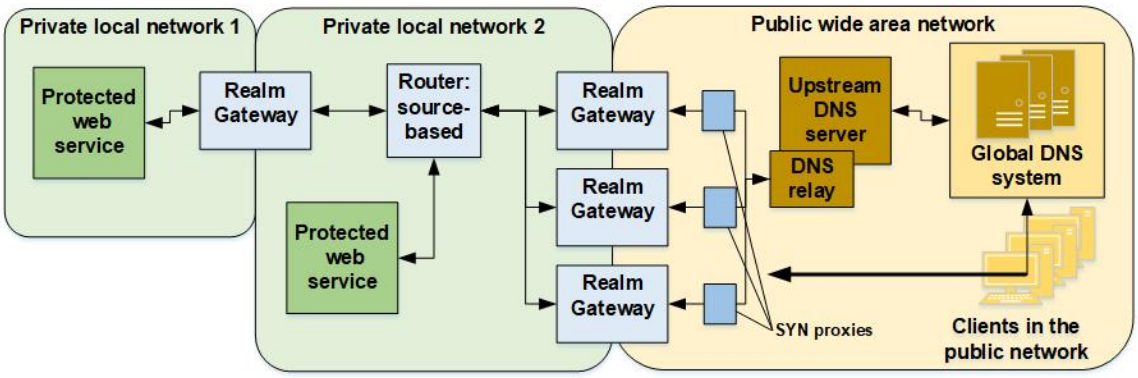

Figure 24: Advanced Realm Gateway setup

NAT which has no state information about the connection and rejects the traffic. In the showcased advanced setup, either the service or a *nested Realm Gateway* would allocate a set of specific IP addresses for the use of each front-line Realm Gateway, where the traffic connected to this set of IP addresses would only use the linked Realm Gateway as a path out. The actual private side routing can be done with a physical router, but it is likely more feasible to do it with a virtual Linux node for example.[123]

In regard to the back-line nested Realm Gateway in 24, it is possible to use multiple layers of Realm Gateway nodes in front of the protected service for more exhaustive DNS client control and mainly to protect against possible threats in the private network. The DNS process with the nested setup starts from the front-line Realm Gateway passing the DNS query to the nested Realm Gateway, as the protected service is marked as carrier grade in the front-line. The response is then relayed back to the client via the front-line, where an access address allocation from the circular pool is done at both gateways. Nesting can be done with or without the load balancing setup, but the source-based routing adjustments become easier with the nested gateway, as the address linking to front-line can be done there instead of changing the actual protected web service configuration.[15]

Originally the nested Realm Gateway setup was designed to work using 1-to-1 or many-to-1 schemes, where a single front-line Realm Gateway with high capacity would serve either 1 or multiple back-line Realm Gateways. It is easy to imagine realistic Realm Gateway deployments, where the front-line Realm Gateways would be managed by ISPs for example and the back-line Realm Gateways could be placed on the ISPs' clients' home routers. During the writing of this thesis, the many-to-many and 1-to-many Realm Gateway relationships were enabled by using source-based routing, the custom DNS relay component and modified Realm Gateway code, but there are other options available to utilize these schemes. For instance, the Realm Gateway software itself could be modified more heavily in order to support the many-to-many and 1-to-many designs from the get-go. Additionally, it may be possible to enable multiple simultaneous front-line Real Gateways without custom DNS relays, if alternative design would be preferable. This could be done with BGP

routers between the Realm Gateway and the public DNS system by using Equal-Cost Multi-Path (ECMP) routing. In this design, BGP routing process would direct DNS queries towards different Realm Gateways by choosing a randomized path from several options in a routing table. There may be issues related to more complex multi-component DNS queries in this case, though, where BGP router software would require custom modules to enable it to keep up some flow state information for example [124].

Finally, in order to deter DoS- and DDoS-attacks, the Realm Gateway does client reputation monitoring based on the client's behavior during the DNS resolution process, where the client identity is linked to the IP source address of the incoming DNS query. The basic idea here is that if the load level is high on the Realm Gateway node, clients who have behaved poorly can be rejected while benevolent clients can still be served. As an additional security monitoring measure, the Realm Gateway can also maintain *whitelists*, *greylists* and *blacklists* about external DNS servers that direct queries towards it. Whitelisted servers are thought be trusted DNS services that can guarantee that they and the connected network infrastructure nearby utilize at least some network security features such as ingress filtering to limit the amount of spoofed DNS queries. It is possible that legal contracts are made with the ISPs that maintain these whitelisted servers to assure certain service level in the security context for example. On the other hand, greylisted DNS servers, which can be most public DNS servers on the Internet, are trusted only to a limited degree. It's possible that malformed or spoofed queries come through them, as the Realm Gateway's controller has just limited influence on these servers. To make Realm Gateway accessible for clients in practice, however, these servers should be entertained at least initially. If servers behave poorly or erratically, they can then be placed on the blacklist, which means that the circular pool resources are denied for these services. It is good to note that these lists can be manipulated dynamically, so that for instance bad behavior from a greylisted server would then lead to it being blacklisted.[125][15]

3.5.3 Custom DNS program structure

The custom DNS server used in conjunction with the Realm Gateway in the thesis tests is a Python 3 software running a combination of UDP and TCP servers in 4 processes and multiple threads, acting basically as a DNS relay. This software was developed by the author of this thesis for the tests in this thesis, and similarly to the Realm Gateway, it uses the dnspython library for DNS message manipulation. The basic purpose of the custom DNS module or DNS relay is to direct DNS queries from the public DNS system or global DNS system to the Realm Gateway and back. During this process, the DNS relay can enable destination randomization, where the DNS queries from the public DNS system are randomized to go to 1 Realm Gateway from a given set of Realm Gateways for load balancing reasons. Additionally the custom DNS can add ECS client subnet data based on the query source IP address to the relayed queries, if this data was not added to the queries on the public DNS

system earlier.[123]

The program structure of the custom DNS module on a base level is shown in Figure 25, where processes 1, 2 and 4 utilize Python's built-in SocketServer library to set up UDP and TCP server functionality. Due to Python software limitations, where threads cannot use multiple processor cores effectively, multiple concurrent processes had to be used in this implementation to make the software scale better when additional system resources could be used. This unfortunately brings about some signaling overhead as data between processes needs to be transmitted through queues.

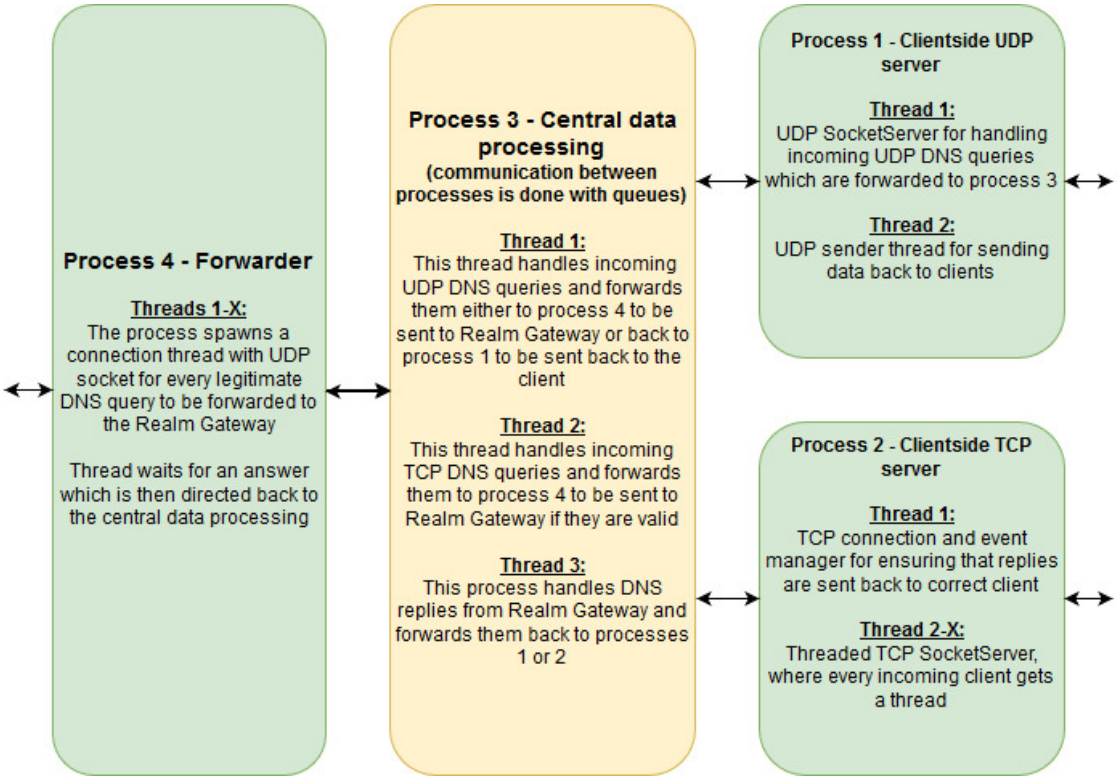

Figure 25: Custom DNS program process and thread structure

The basic handling of incoming DNS queries goes as follows. First the process 1 UDP server receives incoming messages which are directed to process 3, which actually parses the queries and checks what the DNS query contains. The query can then be forwarded either back to the process 1, to thread 2 there, to be sent back to the client or to process 4 if it is to be sent to the Realm Gateway. If the TCP security step is enabled, the client can contact the TCP server, ran by process 2, after it has received the DNS reply with the truncated message flag up. Process 2 can then, in turn, direct messages to process 3 for parsing and forwarding either back to the client or towards the Realm Gateway. With the CNAME security step on, the client does receive the CNAME challenge via the custom DNS eventually and he then needs to answer it, which will result in the allocated address being communicated

back to the client. Note that, as was mentioned before, the communication between the processes happens via queues. This DNS relay implementation also has to store the ongoing CNAME DNS message data in a dictionary data structure in order to connect DNS request and replies correctly.[123]

The custom DNS software has options to set up the Realm Gateway DNS security steps in any combination, and as was noted, to enable target Realm Gateway randomization and DNS ECS use if needed. As is the case with Realm Gateway, the custom DNS does work reasonably well, but it lacks various error-handling functions and advanced memory flushing to handle accumulating CNAME queries, among other things. Ideally, the custom DNS code would be ported to C, as this is a bit more feasible task than porting the more complex Realm Gateway software wholly to C or to some other lower-level programming language. The custom DNS program code is publicly available on the Aalto 5G GitHub and contains instructions and comments which elaborate more on the technical details of the program.[123]

4 Virtual environments and simulating networks

The term *virtualization* in computing and networking refers to creating a *virtual* system, based on some real system, which acts as being the real system. Virtual systems such as virtual computing hardware and virtual data storage have many practical uses, where the creation, maintenance and handling of security issues with virtual systems can be much more streamlined and cost-effective for example than dealing with actual physical hardware. The key concepts with virtualization in regard to this thesis are the Virtual Machine (VM) and virtualized environment, where they usually are a virtual, functioning software representation of some actual computer running a specific OS.[126]

4.1 Virtual machine concept

The key terms with VMs, virtual environments and hardware virtualization are *the guest*, *the host*, and the software that will usually run the virtualization process called *hypervisor*. An example structure of a VM setup is shown in Figure 26, where the host is the actual computer system running the hypervisor that, in turn, runs the guest VMs that may run different OSes than the host. The hypervisor, such as Oracle's VirtualBox [127], creates a system environment for the guests where they see a software abstraction of the underlying hardware but they think it is the real deal. This means that the software running on the guest OSes is basically isolated to a sandbox environment, where it can access the actual hardware resources in a very limited and controlled manner.[126]

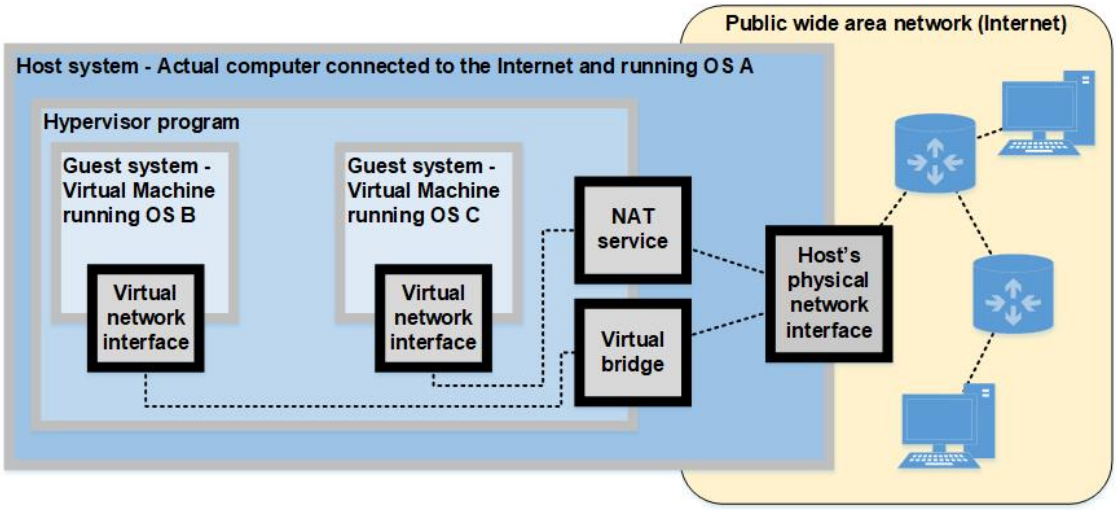

Figure 26: Virtual Machine setup example

For network connectivity, the hypervisor will usually link the guest systems to the host's physical network interface via middle-man device. The common options include setting up a NAT service between the guest and host, where the guests would be in a private network space, or setting up a virtual bridge device, where the host

interface is bridged directly to the guest's network connection. In the latter case, the guest and host share the same network addressing space, and if the host would be connected to the Internet directly, so would the guest be, with a global IP address allocated to it.[128] Using bridges, be it virtual or physical, refers to connecting network nodes together via link layer bridge device that will enable the connected devices to share the same Internet layer network. Basically the bridge has its own MAC (link layer) address, sharing this address space with the connected devices, and it can then relay data between the nodes on either side of the bridge, simulating a direct connection in the Internet layer.[129]

It was implied earlier that utilizing system virtualization with VMs offers many benefits, especially with program testing in the context of this thesis. The sandbox environment makes running problematic software more secure, as it cannot access the host directly. The isolation and easy installation and re-installation of the guest systems and their OSes also makes it quicker and more straightforward to test complex software configurations, as errors caused by program conflicts can be minimized and the whole system can be built from scratch many times over if some major error brings the guest down for good. Finally, it is far more cost-effective to test various basic network setups with VMs, as setting up and utilizing actual physical routers instead of virtual routers would be far more expensive and would take more time. One major downside with VMs is the resource use overhead of the hypervisor program, though, which has to be taken into account if software performance tests are done in virtualized environments.[126]

4.2 Linux containers

When VMs are run with hypervisor, the processing overhead can be a problem because the hypervisor needs to manage the hardware-level abstraction for the guests. Another option is to use more light-weight virtualization, if the guest OS can be of the same basic architecture than the host, as is the case with *Linux Containers* (LXC). LXCs are essentially Linux guests running on a Linux host, where the guests are allocated their own process space, CPU resources, memory and disk storage. With Linux, it is possible to do *chroot* operations, where the root location of the process and its child processes is changed, which then limits what these processes can access [130]. The container's own principal Linux OS processes and subsequent standard applications are run in this manner with Linux *namespaces*, so they are practically isolated [131]. Other computing resources for the container can then be partitioned with the use of Linux *cgroups*, where the processes of a guest can only use what is specifically allocated for them [132]. Also, It is good to note that LXCs are not VMs in the strict sense but virtualized environments instead, even if they aim to do the same thing.[133]

Major boon with LXCs is that the hypervisor software that manages the containers can be very limited, which reduces the processing overhead of running guests drastically. Generally, the LXC manager software will do limited monitoring on the

guest processes so they don't do anything unexpected, but overall, the guests can access host hardware resources in a more direct manner, as they share the same OS functionality. Additionally, LXC environments are quicker to install and remove because there is no need to abstract the hard disk storage for the container OS. A new container can just split a part of the underlying host's file system for itself instead. The downside with LXCs, in contrast to proper VMs, is the breaking of the container isolation. It is possible that a user within the container could gain access to underlying, restricted system resources or would be able to enter the host's OS file system space by a software bug for example. To help defending against this threat, it is important to limit who can access the guest systems, especially as a root. It is also wise to run the LXCs in unprivileged mode, so that if a root user in the guest escapes to the host, his identity is not root anymore which limits his capabilities.[133]

4.3 Linux containers and network virtualization

The limited LXC hypervisor offers similar network connectivity to the containers than more complex VM hypervisors. It would be possible to use a NAT service between the host network interface and the containers which was one option in Figure 26, but more suitable option for hosting publicly accessible web services in the containers would be to use a virtual bridge to connect the host's physical interface to the container's network interface. An example of this bridged setup is shown in picture 27, where the network topology consists of a public network and two private networks. The public network can be directly accessed by the host and the Linux container 3, whereas private network 1 and 2 are only accessible by the corresponding Linux container 3 interfaces and by the Linux containers 1 and 2 respectively.[133]

The key part of enabling networking with VMs and with LXCs is the creation of virtual Ethernet devices. These devices have actual MAC addresses in the host system, so data traffic can be directed to them by the hypervisor and by the host OS. The virtual network interface of a container or VM would be a combination of this virtual Ethernet device and an abstraction of a physical network interface within the container or VM. In Figure 27, each virtual network interface with each container is connected to an abstracted, standard network interface in that container in this manner. As virtual network interfaces have MAC addresses, they can then be connected to virtual bridges which simulate direct connections between the bridged nodes. This would mean that if private IP addresses of a suitable scope are allocated to Linux containers 1 and 2 for example, they can reach each other with IP, as they would seem to be on the same network segment. Going beyond specific network, such as from private network 1 to private network 2, would require routing procedures from the intermediate container 3, though.[133]

As data traffic on the Internet relies on IP, utilizing LXCs, virtual bridges and virtual network interfaces offers a good way to observe Internet layer, transport layer and

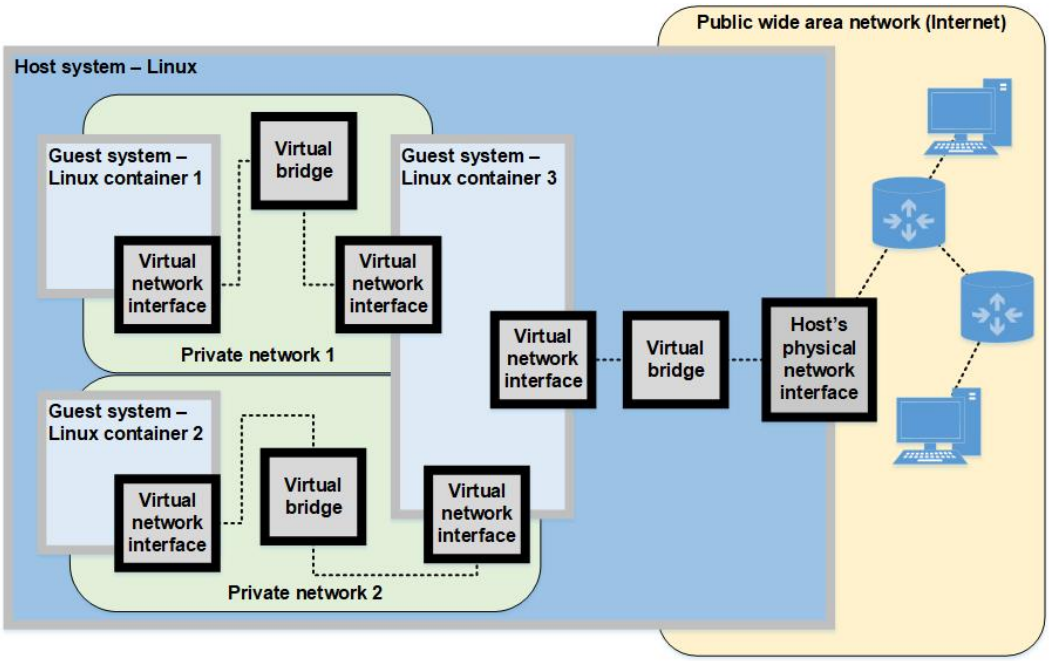

Figure 27: Networking with Linux containers

application layer network behavior in a contained environment. For example, a high-level web service could be run in isolation within a container to limit possible software conflicts and the incoming data traffic through the virtual network interface would be indistinguishable from traffic from an actual physical interface. For the last note, there is one minor downside. Even if the general container management with LXC is very quick, there can be some operational overhead with virtualized data input and output operations (such as networking), stemming partly from the fact that physical hardware can do actual circuit-level optimizations. For most network traffic, this is not a big problem as the differences come more apparent only with a quick succession of very small data packets.[133][134]

5 Testing a secure web service

To find out if the Realm Gateway software and customized DNS resolution process could be used to help defend against DoS- and DDoS-attacks, three network testing setups were created for this thesis. The first setup was run purely in Ubuntu Linux VirtualBoX VM on a Windows host computer, where the network nodes were LXCs in the Ubuntu VM, connected by virtual bridges. The second setup used a set of 6 Ubuntu Linux VMs that were hosted on a cloud service, where each VM had a real, global IPv4 address. In the first setup with 1 Ubuntu VM hosting the whole simulation setup, the LXCs within acted as protected web services, Realm Gateway nodes, accompanied DNS systems, and clients for the protected services, where the clients generated data traffic towards the service. With the cloud service setup, different VMs in the cloud acted as clients, DNS systems, Realm Gateways and protected services behind them, where the LXCs were set up for network nodes in the respective VM, if necessary. Also, with the cloud, the client nodes generated either legitimate or hostile data traffic towards the service, depending on the test. The third and final testing setup was a set of 4 physical computers running Ubuntu Linux, all connected to the same private network using a physical switch. In this last setup, one computer would host the LXCs for the Realm Gateway and the protected service behind it, another computer would act as the DNS system and the remaining computers would act as clients. The relevant computer hardware and software specifications for the aforementioned setups, such as Central Processing Unit (CPU) information and system memory amount are presented in Table 5.

In regard to the actual testing, the first set of tests involved validation procedures to ensure that the Realm Gateway and the accompanied DNS and routing systems would be able to handle common data traffic between legitimate clients and the protected service. Further tests concentrated on measuring system responsiveness and computational resource use under varying traffic loads. The final tests then measured system performance when it was targeted by various kinds of DoS- and DDoS-attacks. Note that when deciding on the IP addressing for the test scenarios, Realm Gateway circular pool needed to have at least 3 IP addresses allocated to it for the system to work properly, so this was taken into consideration [122]. The 4 testing categories are elaborated further below:

1. System validation for testing that protected service could be reached through the Realm Gateway by legitimate clients utilizing UDP and TCP communication and the simulated DNS system.

2. System delay testing for observing how long it would take to resolve DNS queries directed to the Realm Gateway DNS server and how long it would take for the client to contact the protected service through the Realm Gateway.

3. Measuring CPU and memory use for Realm Gateway node and the custom DNS node when they are handling varying levels of incoming data traffic.

Table 5: Computer and software specifications for tests

System	**Hardware**	**Software**
Desktop host PC for test setup 1	CPU: Intel 4-core, 3.2 GHz Memory: 16 GB	OS: Windows 10 Hypervisor: VirtualBox 5.2 Guest: Ubuntu 16.04 VM Guests's hypervisor: LXC 2.1 LXCs: Ubuntu 16.04
Cloud VM for test setup 2	CPU: Intel 4-core, 2.6 GHz Memory: 8 GB	OS: Ubuntu 18.04 Hypervisor: LXC 3.0 LXCs: Ubuntu 16.04
Desktop host PC for setup 3	CPU: Intel 4-core, 3.8 GHz Memory: 16 GB	OS: Ubuntu 18.04 Hypervisor: LXC 3.0 LXCs: Ubuntu 16.04
Laptop host PC for setup 3	CPU: Intel 4-core, 2.0 GHz Memory: 6 GB	OS: Ubuntu 18.04 Hypervisor: LXC 3.0 LXCs: Ubuntu 16.04
Client laptop PC 1 for setup 3	CPU: Intel 2-core, 1.5 GHz Memory: 4 GB	OS: Ubuntu 18.04
Client laptop PC 2 for setup 3	CPU: Intel 2-core, 1.8 GHz Memory: 3 GB	OS: Ubuntu 14.04

4. System delay testing during various DoS- and DDoS-attacks, where attack traffic would be generated towards the Realm Gateway and one would observe the delays this causes for legitimate users.

The tests within each of the four categories are presented in more detail in the following subsections where the respective test setup network topologies are also shown. The results of these tests are then presented and discussed to some detail at the end of each test category presentation. As a general note, all LXC containers had unlimited access to underlying system resources with all tests. Additionally, DNS system components were all Python programs, and Python version 3.7 was used in running them as well as additional networking scripts with services and clients if necessary. The Realm Gateways and SYN proxies were run with Python 3.5.

Finally, In regard to delay results presented later in this chapter, the figure format is similar to all tests, where the bars in each graph show the incurred delay averages, with different bars in a cluster denoting different setups which are also explained in the figure legend. The dot/circle plot on these graphs then denotes the delay median, and the cross plot denotes connection or name resolution failure percentages where the value can be seen on the rightmost y-axis. Note that if cross plot is not present, that means that no connection or resolution failures were noted. Resource use figures are of similar format, but there, the y-axis values are spent CPU time and memory

use average instead.

5.1 Validation testing

Validation testing was done using the test setup 1 from Table 5, where the whole system of guest LXCs and virtual bridges in a VirtualBox Linux Ubuntu VM was hosted on a Windows PC. These tests involved utilizing 3 different network topologies, with increasing routing complexity. The validation test setup 1 is shown in Figure 28, where the blue squares note LXC nodes running either Realm Gateway software, SYN proxy scripts or DNS servers, or acting as a router or as a client with the respective communication tools to reach the protected service. The containers were connected with virtual bridges and the network was separated to two distinct segments: the public wide area network, simulating global Internet, and the private network, protected by the Realm Gateway.

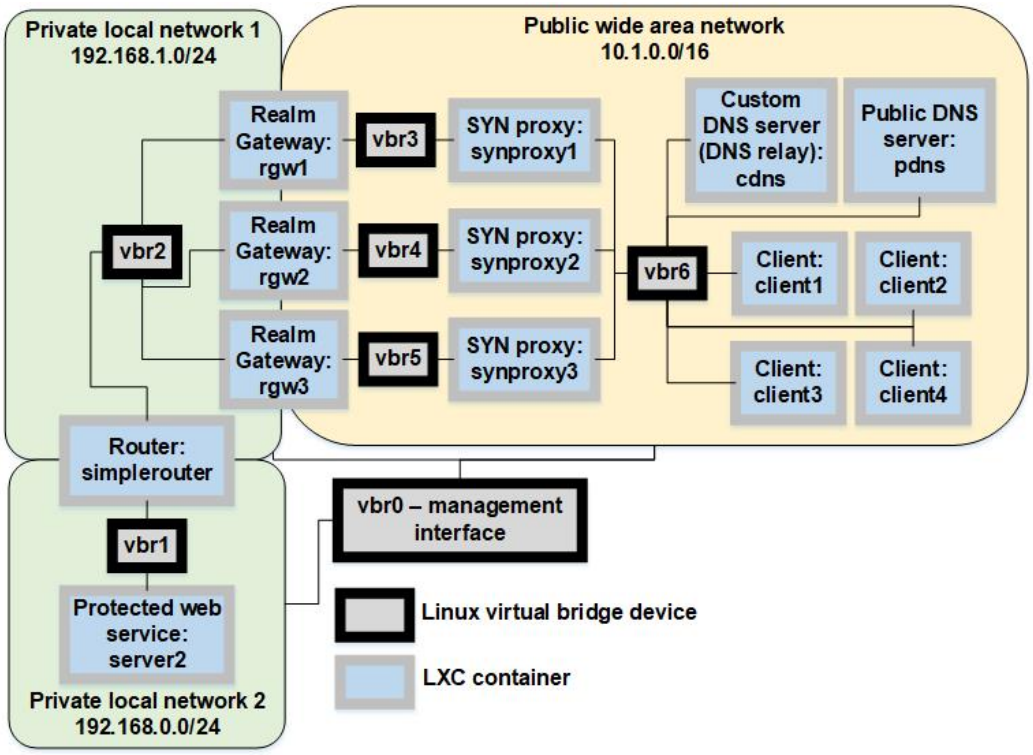

Figure 28: System setup with simple network topology for validation test 1

Using the setup in Figure 28, the clients could reach the protected service by first contacting the public DNS server emulating the global DNS system, in order to resolve the IP address for the service. The public DNS forwarded this query to the custom DNS which then forwarded it to the Realm Gateway. If the client passed the TCP and CNAME DNS steps successfully, the Realm Gateway then allocated an IP address for the client from its pool of 3 public IP addresses meant for pass-through

traffic. This address was then communicated back to the client via the DNS system so that the client could contact the service through the Realm Gateway's NAT, connected to the allocated public IP address.

With multiple Realm Gateways placed simultaneously in front of the private network, the client could reach the protected service through each Realm Gateway. This required some additional routing procedures on the private network side, as generally nodes would only have just 1 network gateway. First, the custom DNS component randomly chose the Realm Gateway for which to direct the queries for load balancing reasons. Then, the selected Realm Gateway resolved a specific protected service IP address connected to it for the DNS process. This basically meant that the protected service ran 3 different IP addresses, each of which was linked to just one of the 3 Realm Gateways. Finally. the traffic back from the protected service to the client did utilize source-based routing in the router on the private network, which directed traffic from a specific service IP to the respective Realm Gateway, so that the traffic reached a gateway that had the respective connection state up. Source-based routing could be done with just basic Linux kernel routing functions and iptables, as they make it possible to mark packets based on their source IP addresses and then direct these packets to a specific gateway address.

More advanced validation test setups are shown in Figure 29. With setup 2, the clients tried to reach protected service in node server2 in the private network 2 and with setup 3, there was a nested, additional Realm Gateway behind the first line of Realm Gateways, where clients tried to reach protected service running on node server1 on private network 1. The public network side with these setups worked identically to setup 1 but there was additional routing complexity on the private network side to emulate Realm Gateway deployment scenarios where it would need to connect to the protected service through public internet.

To elaborate on routing here, as was the case with setup 1, the node router2 would do source based routing in order to enable the use of multiple front-line Realm Gateways simultaneously. With setup 2, the front-line Realm Gateways were each connected to a separate IP address on the node server2. With the nested Realm Gateway in setup 3, each of the front-line Realm Gateways were actually connected to specific IP address from nested Realm Gateway's the circular pool. In setup 3, the nested Realm Gateway would allocate an IP address from the pool based on the DNS query source address, as this would indicate which front-line Realm Gateway sent the query. Note that the nested Realm Gateway required minor modifications to the default code base to enable it allocating certain addresses from its circular pool based on the source of the incoming DNS query. Basically, instead of the standard starting procedure, it created 3 instances of circular pool in this case, where each instance was linked to 1 specific front-line gateway.

To enable secure transmission over public Internet on the private side, the network topology was set up with BGP communities in Figure 29, which essentially meant

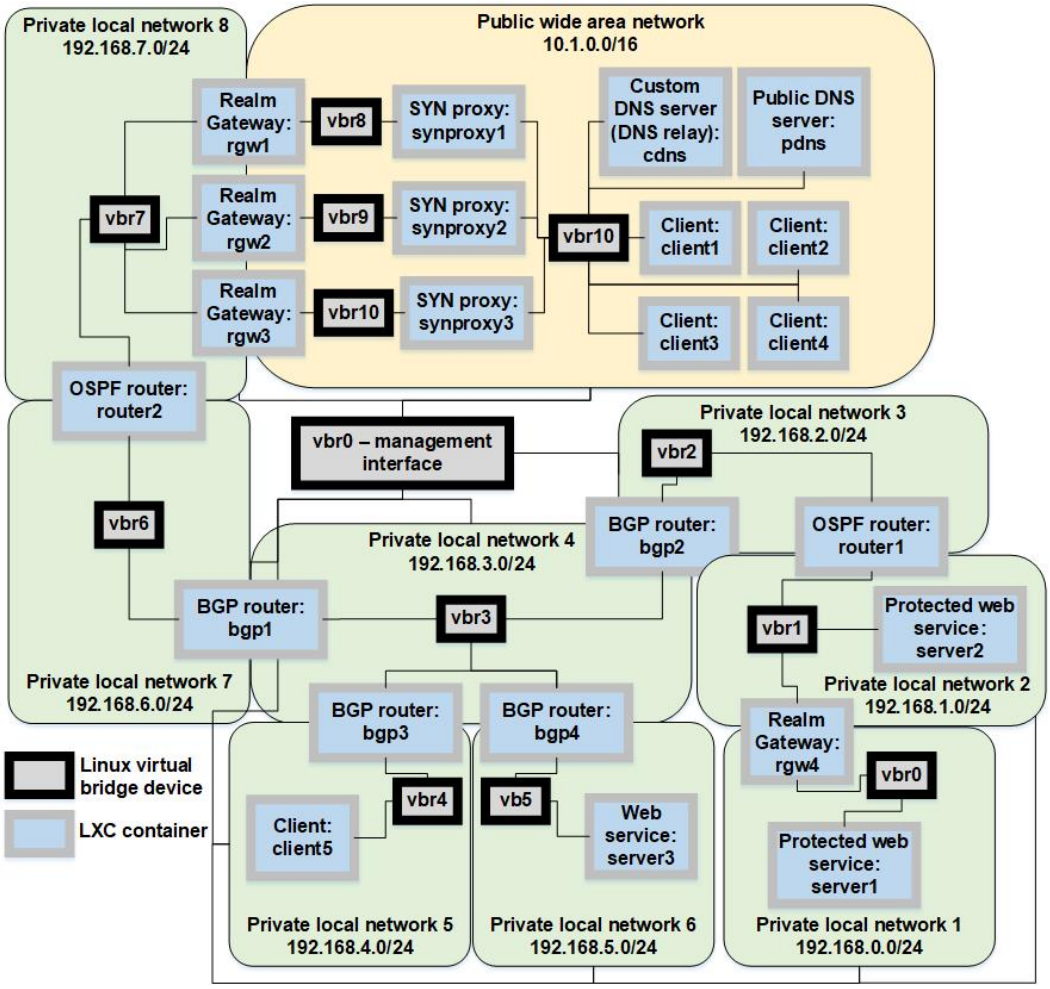

Figure 29: System setup for validation tests 2 and 3 with complex routing

security by detachment. The private networks 7 and 8 were thought to be trusted networks connected to the Realm Gateway, where the BGP router bgp1 was a trusted AS border router for these network segments. A shared BGP community was then created between this router and the BGP router bgp2, acting as a AS border router for trusted private networks 3, 2, and 1. Using this BGP community, the addresses from networks 1, 2, 3, 7 and 8 would only be advertised within the community, which meant that the ”untrusted” BGP routers bgp3 and bgp4 didn't have knowledge about these addresses. This also meant that entities on ”untrusted” networks 4, 5 and 6 didn't have access to these addresses, hence the service detachment, if the networks 4, 5 and 6 could be thought as global Internet residing between the trusted networks in a very simplified sense. In order to do actual BGP and OSPF routing, the routing containers in the private network ran Quagga routing suite [42], where the OSPF routers would use the nearest BPG routers as a gateway to wider network, and where the BGP routers were all BGP peers with the added BGP community limitations.

The validation tests themselves were relatively straightforward as the listing below shows:

1. Validation test 1 for the Realm Gateway setup with multiple access gateways
   (a) Test for client access to protected service running UDP server through Realm Gateway
   (b) Test for client access to protected service running TCP server through Realm Gateway
   (c) Test for client access to protected service running SSH server through Realm Gateway
   (d) Test for client access to protected service running HTTP server through Realm Gateway
2. Validation test 2 for the Realm Gateway setup with multiple access gateways and advanced routing - Same access tests (a) to (d) for this as for validation test 1
3. Validation test 3 for the Realm Gateway setup with multiple access gateways, advanced routing and additional nested Realm Gateway - Same access tests (a) to (d) for this as for validation test 1

The protected service listened for incoming UDP and TCP connections with Linux *Netcat* program [135]. The SSH connection to the service was done by utilizing the Linux *OpenSSH* server and the built-in Linux SSH client [136]. The HTTP web page retrieval was done with *Wget* tool [137], where the protected service hosted web pages using *NGINX* server software [138]. Note that the HTTP reverse proxy could have been used to handle HTTP traffic with the validation tests but due to the simplicity of the NGINX default page that was retrieved during the tests, and due to technical issues with enabling the reverse proxy process, HTTP traffic was directed through using the normal Realm Gateway DNS resolution and circular pool access address allocation.

5.2 Validation testing results

All presented validation tests were done successfully, from 1-a to 3-d. With validation tests 1-a to 1-d, with Figure 28, all the clients could contact server2 with all presented protocols while the custom DNS server randomized the Realm Gateway during the name resolution process. With validation tests 2 and 3 (from 2-a to 3-d), with Figure 29, all the clients on the public wide area network could connect to server 2 and server 1 respectively with all presented protocols, again with the front-line Realm Gateway randomization. Additionally, with tests 2 and 3, it was noted that client5 and server3 could not access private networks 1, 2, 3, 7 and 8.

In principle this shows that the Realm Gateway supports at least the most common network protocols successfully and works with the basic network secure service setup of hosting a web service that would be accessed with TCP utilizing at least some encryption and access control methods such as SSH. Additionally, the validation test results show that it is possible to utilize multiple simultaneous front-line Realm Gateways to do load balancing, although this requires additional routing setup in the private network. Finally, it is possible to detach private network segments for the use of Realm Gateway and protected service with BGP communities to enable secure data transfer from the Realm Gateway to this service through untrustworthy networks. This, however, requires a chain of trustworthy BGP routers that starts at the gateway's immediate network or AS border and continues to the service's immediate network or AS border.

5.3 System delay testing

For system delay testing, resource use measurements and for the majority of the DoS- and DDoS-related tests, the cloud setup shown in Figure 30 was used. The procedure for the clients to connect to the protected services worked in the same manner as with validation tests. To conserve computing resources with the VMs hosting the Realm Gateways, the load balancing of the front line Realm Gateways was emulated by setting up the protected service and nested gateways as two instances, where ideally they would have been just one instance each, located on a shared private network behind both the Realm Gateway 1 and 2, as was the case with validation testing. Essentially, this setup would still showcase the effects of load balancing on the front-line Realm Gateways, though. In a similar manner to the VM hosts for the Realm Gateways, one VM in this setup hosted the custom DNS server running on a LXC, protected by the SYN proxy. In regard to network connectivity with the cloud setup, network delay for all VMs and LXCs to other entities in the cloud and in the test setup was less than 1 millisecond.

The basic delay tests utilized only 1 Realm Gateway and respective protected service. All tests were done with 3 different security setups, where 1 setup would have Realm Gateway and custom DNS just using normal DNS, another setup would make custom DNS ascertain client legitimacy with DNS response with the truncated message flag up and subsequent expected DNS TCP reply and final setup with both the TCP step and CNAME challenge enabled. With the delay tests here being done for legitimate clients, the public DNS was set up to do the TCP step and CNAME step on behalf of the clients, as for example, BIND DNS servers could be configured to automatically complete the aforementioned steps with recursive queries, if they can forward the CNAME queries correctly back to the custom DNS and Realm Gateway. All tests here used a Python network script for the client to send and receive DNS queries and answers and subsequently contact the service through the Realm Gateway. In delay tests, the custom DNS SYN proxy was also disabled. The tests themselves were done in the following manner:

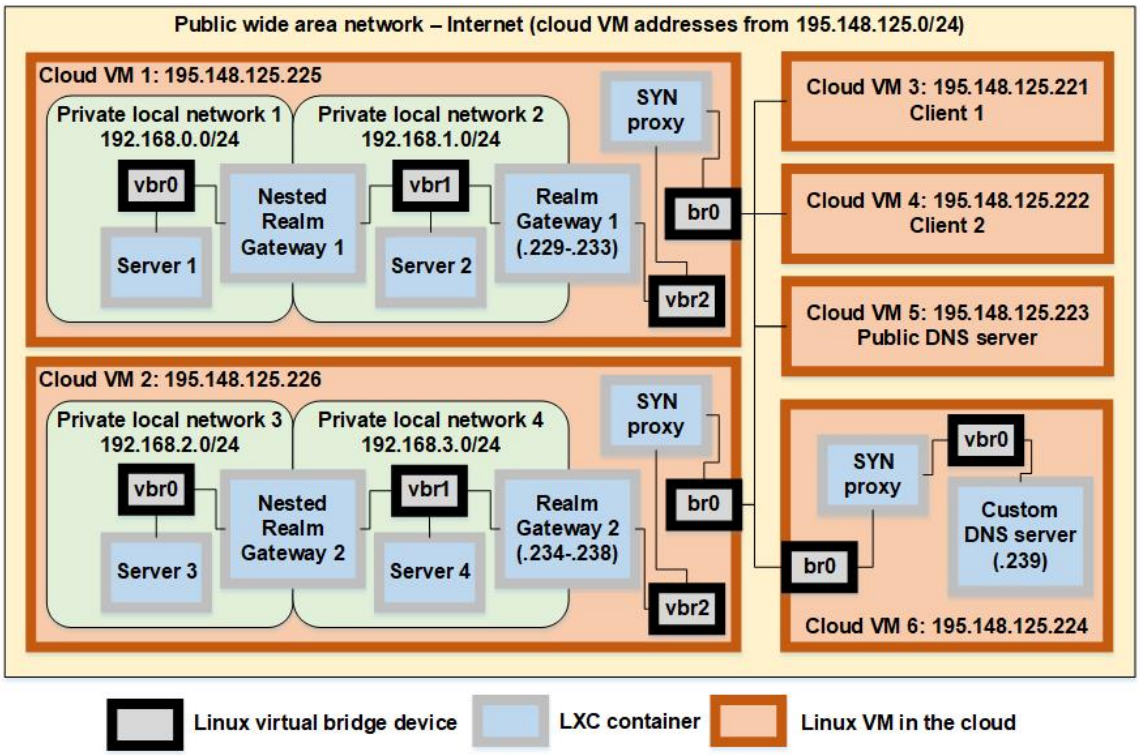

Figure 30: Cloud service system setup

1. System delay test 1 for measuring the DNS name resolution time between the client sending a DNS query to the public DNS server and then receiving a valid answer from the Realm Gateway (with custom DNS server acting between the public DNS server and Realm Gateway). The noted result is the average and median from all successful attempts while discarding the highest and lowest 10% of the delay set. Additionally, the number of failed queries is also to be noted with the client doing no re-tries for an attempt. The client query amount per second is varied. To clarify, the VM client 1 would contact Realm Gateway 1 in Figure 30 here.

   (a) Test the above with no added delay.

   (b) Test the above with added 5 millisecond delay for the client 1 and the custom DNS server network interfaces using Linux traffic control to emulate more challenging network environment.

2. System delay test 2 for measuring the time between client 1 sending a DNS query to resolve protected service IP address and actually contacting the service. The service was a Python TCP echo server running on server 2. Otherwise the tests here were similar to test 1.

   (a) Test the above with no added delay.

   (b) Test the above with similar added delay to test 1-b.

3. System delay test 3, similar to test 1 but now with a nested Realm Gateway. In this case the client 1 would contact server 1 through the two gateways.
   (a) Test the above with no added delay.
   (b) Test the above with added 5 millisecond delay to the client 1's network interface, to public DNS server's network interface and to the Realm Gateway 1's private side network interface, in a similar manner to tests 1-b and 2-b.
4. System delay test 4, similar to test 2 but now with added nested Realm Gateway (here client 1 would try to contact server 1).
   (a) Test the above with no added delay.
   (b) Test the above with added 5 millisecond delay in a similar manner to test 3-b.

Mainly due to server capacity limitations from Python implementation, which are discussed more in Appendix C, care had to be taken to set the incoming query rates to relatively low levels compared to commercial DNS server software such as BIND. To avoid major application level server congestion, critical services usually employ some kernel level rate limiting, which can be done with iptables for the LXCs running the Realm Gateway. This limit was set to 200 queries per second, and tests here and beyond would be within this scope to get more accurate information on the application layer behavior, as going far above the aforementioned limit would induce dropped packets due to firewall rules or filled link buffers.

5.4 System delay testing results

Generally for the delay tests, the Realm Gateway started to suffer notable resolve or connection failure rates when TCP and CNAME DNS security features were turned on and the incoming query rate went above 50 queries per second. This problem became more apparent with the nested Realm Gateway cases, as the front-line Realm Gateways need to use more resources to query the nested Realm Gateway about the service address while simultaneously maintaining system security with the public network clients and DNS systems. These were the reasons to limit tests to the rates between 1 and 75 queries per second with non-nested setup, and between 1 and 25 queries per second with the nested setup. It is worth noting here, that this doesn't indicate that the system couldn't handle DoS- and DDoS-attack traffic of similar or far larger magnitude, as handling legitimate clients uses lots of server resources due to the address allocation process, whereas just discarding illegitimate UDP DNS queries from the get-go with UDP flood is far simpler. It is also the case that when provisioning security systems, the capacity to handle attack traffic is often set to much higher level (10-fold or even 100-fold) compared to the capacity to handle simultaneous legitimate clients. This difference becomes more drastic, if the system

in question wouldn't expect flash mob-type scenarios.

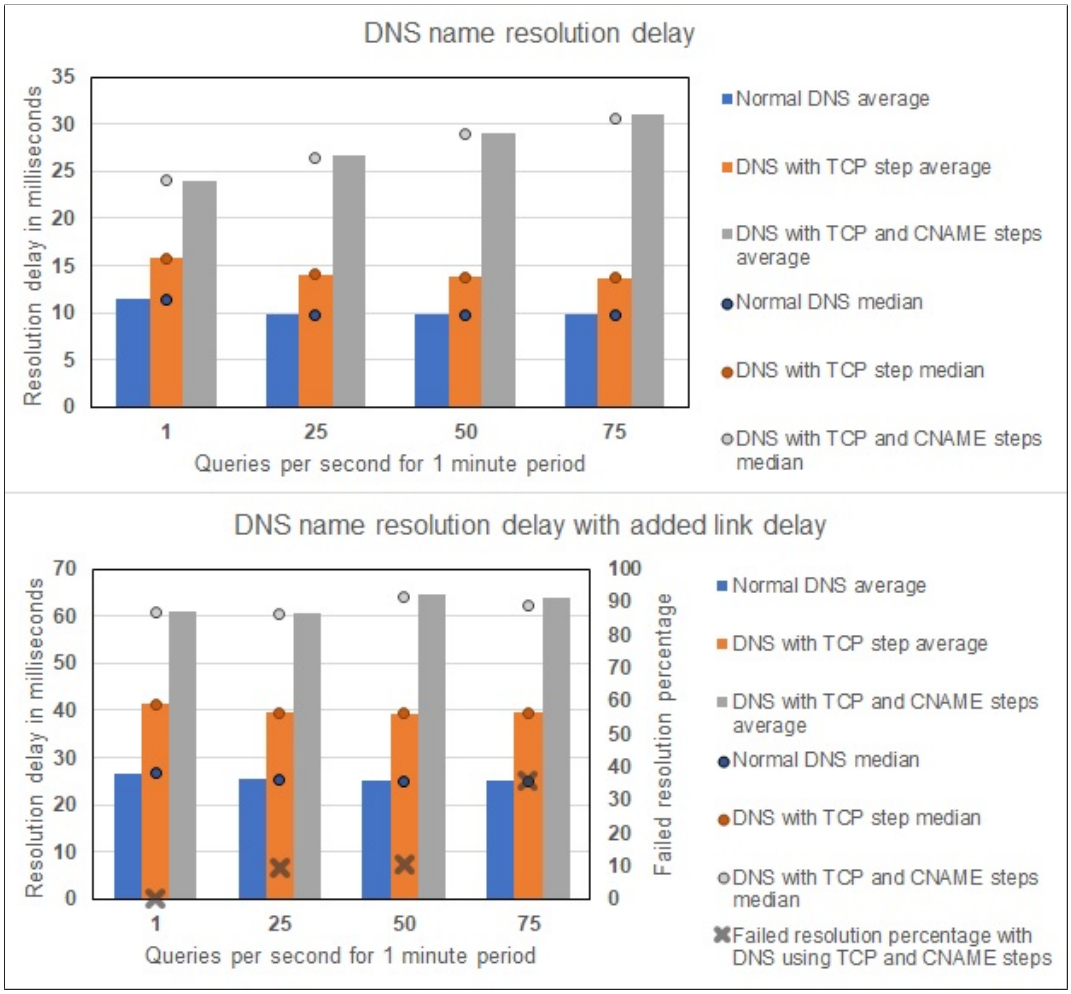

Figure 31: Delay test 1-a and 1-b results for DNS name resolution delay with non-nested Realm Gateway setup

The results of delay test 1-a are shown in Figure 31, in the upper graph. It can be seen there that adding security steps to the DNS process will incur additional delays, but it is another question if these delays are dramatic enough to make the system unfeasible to use. With ideal circumstances, with minimum delay, the name resolution delays stay at very low level, with the whole process taking around 30 milliseconds at maximum, even with the higher loads. This is well within the boundaries of Windows or Linux OS DNS resolver timeouts, which are 1 second for the first attempt with Windows, and 5 seconds for the first attempt with Linux systems, generally by default [139][140].

If simulated delay is added to the system, some problems come about, though, as can be seen in the lower graph in Figure 31. In this case, the client's and custom DNS's network interfaces had additional 5 millisecond delay, which caused some

failed resolution attempts at higher query loads when both TCP and CNAME steps where enabled. It is likely that this worsens as the network delays increase, but ideally, at least the custom DNS component, Realm Gateway and the protected service would be near each other to limit delays between them. Successful resolution attempts were still relatively fast, though, as was the case with test 1-a. If DNS resolution attempts fail, it is not as big of a problem as both Linux and Windows DNS resolution processes do multiple attempts to resolve the domain name IP address which can increases the success rates to connect to the actual service dramatically.

Additionally, it is important to note that due to the limited time-frame of Realm Gateway address allocation upon successful DNS resolving process, it is not feasible to set up Realm Gateway to a network topology where it takes 100s of milliseconds for the client to contact the DNS system components and the Realm Gateway through SYN proxy, or through multiple SYN proxies if the custom DNS has a separate SYN proxy. As networks with these kinds of dramatic delays would make even phone discussions problematic, this is not that big of an issue, as lesser delays should be expected for most deployment scenarios.

The results of delay tests 2-a and 2-b are shown in Figure 32, in the upper and lower graph respectively. From these, the same limitations become apparent as with delay test 1, with the added network delay. It is noteworthy here that the Realm Gateway itself doesn't cause large delays for the pass-through traffic, and the major hold-up is the DNS process, as expected.

Delay test 3-a and 3-b results are shown in Figure 33, again, in the upper and lower graph respectively. With nested Realm Gateway, the main difference to previous delay tests is the decreased capacity. It was noted, especially with the TCP and CNAME steps on, that the system had problems coping with traffic rates of 50 queries per second and up as the front-line Realm Gateways had to do additional work when they forwarded incoming queries to the nested Realm Gateway. Delay for successful resolve attempts stayed reasonable, though, as it was well below the 1 second threshold, even with all the DNS security steps on.

Finally, the results of delay tests 4-a and 4-b are presented in Figure 34, in the upper and lower graphs. The results here mirror test 3-a, 3b and the non-nested service connection delays, as the Realm Gateway itself doesn't affect TCP traffic passing through that much, which is the case even with multiple consecutive Realm Gateways on the traffic path. With nested Realm Gateways, there is additional security for the protected service, especially if elements of the private network side are not trustworthy, but the latter tests here show that there is a trade-off with capacity with the front-line Realm Gateways as they have to do more work. As a concluding note about delay tests and the somewhat limited legitimate query handling capacity, it is good to understand that Realm Gateway is not really meant to be an all-purpose security solution for any kind of web service, but it is meant to protect critical services that have limited legitimate user base instead, where valid query handling rates of roughly 10 queries per second should suffice quite well in

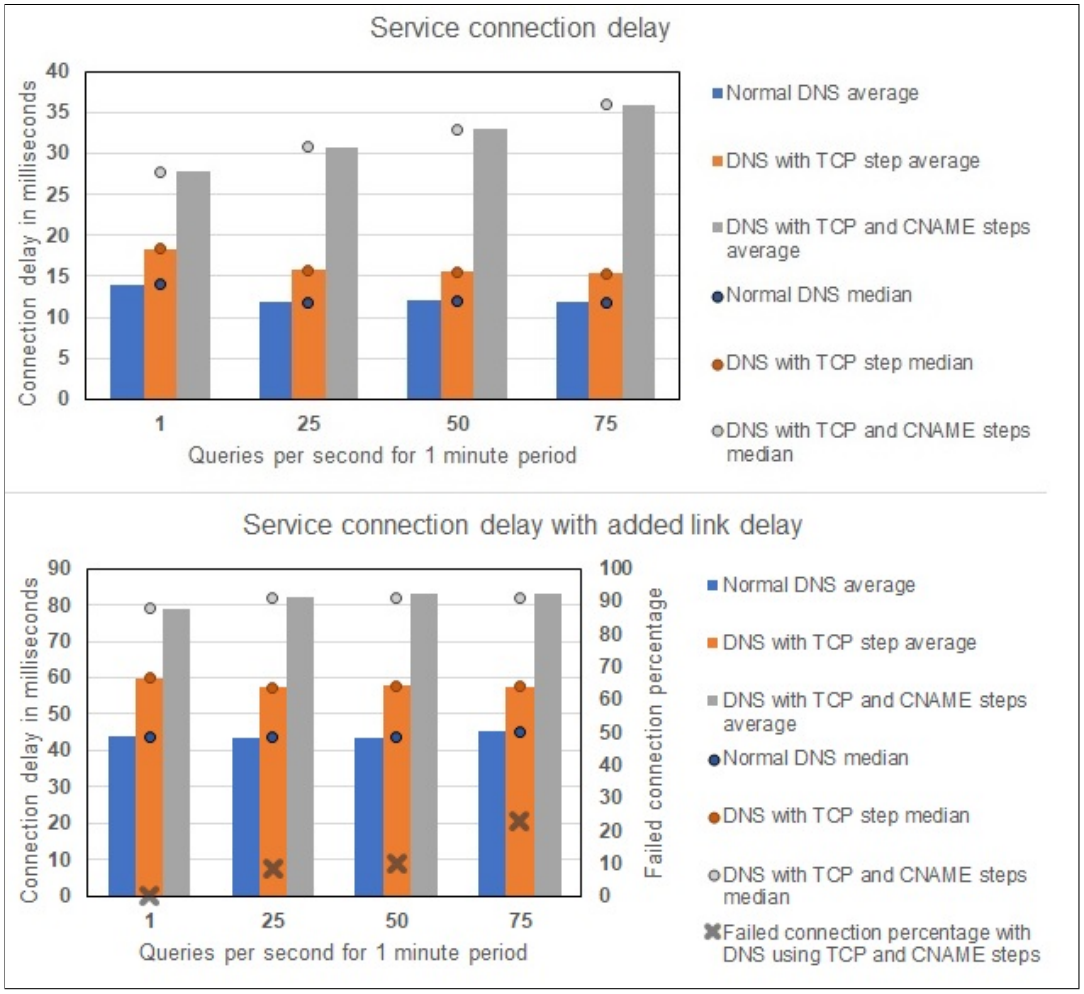

Figure 32: Delay test 2-a and 2-b results for delay for connecting to the protected service with non-nested Realm Gateway setup

most cases.

5.5 Computational resource use measurements

The resource use measurements were done to test how much CPU time and system memory Realm Gateway 1 and custom DNS server would use during the situation similar to delay test 2-a with the varying query load, also with varying the DNS security setups. The respective container's base level resource use was also to be included. Basically, the tests were as follows:

1. Resource use measurement 1, where container's used CPU time is noted for 20 seconds during a period of varying incoming query load. Take average from three separate measurements.

   (a) Test the above for the LXC running the custom DNS.

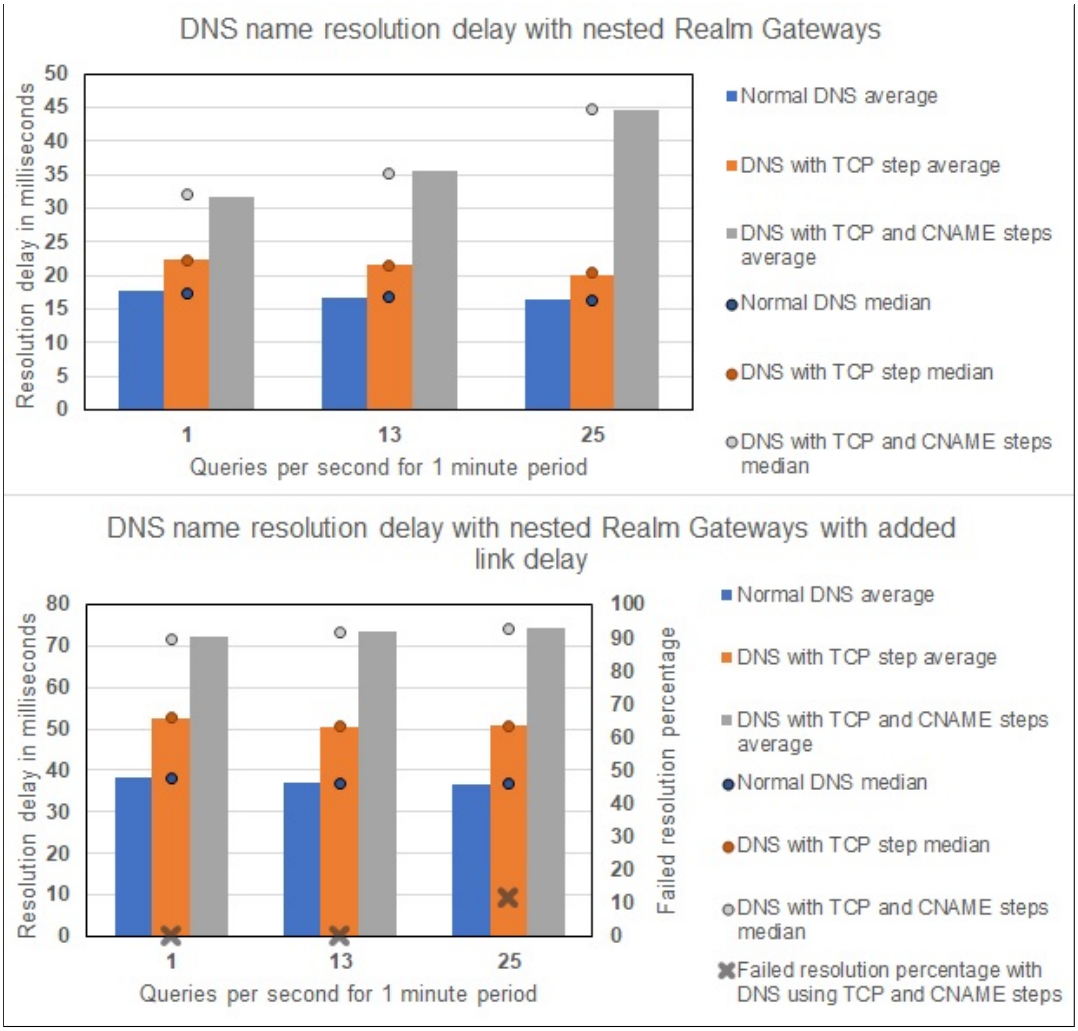

Figure 33: Delay test 3-a and 3-b results for DNS name resolution delay with nested Realm Gateway setup

(b) Test the above for the LXC Realm Gateway 1.

2. Resource use measurement 2, where container's used memory average is noted for 20 seconds during a period of varying incoming query load. Take average from three separate measurements.

   (a) Test the above for the LXC running the custom DNS.

   (b) Test the above for the LXC Realm Gateway 1.

5.6 Computational resource use measurement results

The resource use measurements were done while the custom DNS and Realm Gateway were under a similar load as with the delay tests in non-nested cases. Here the results show how much CPU time the LXC running either custom DNS or Realm Gateway took during the 20 second measuring period, under varying legitimate query rate levels.

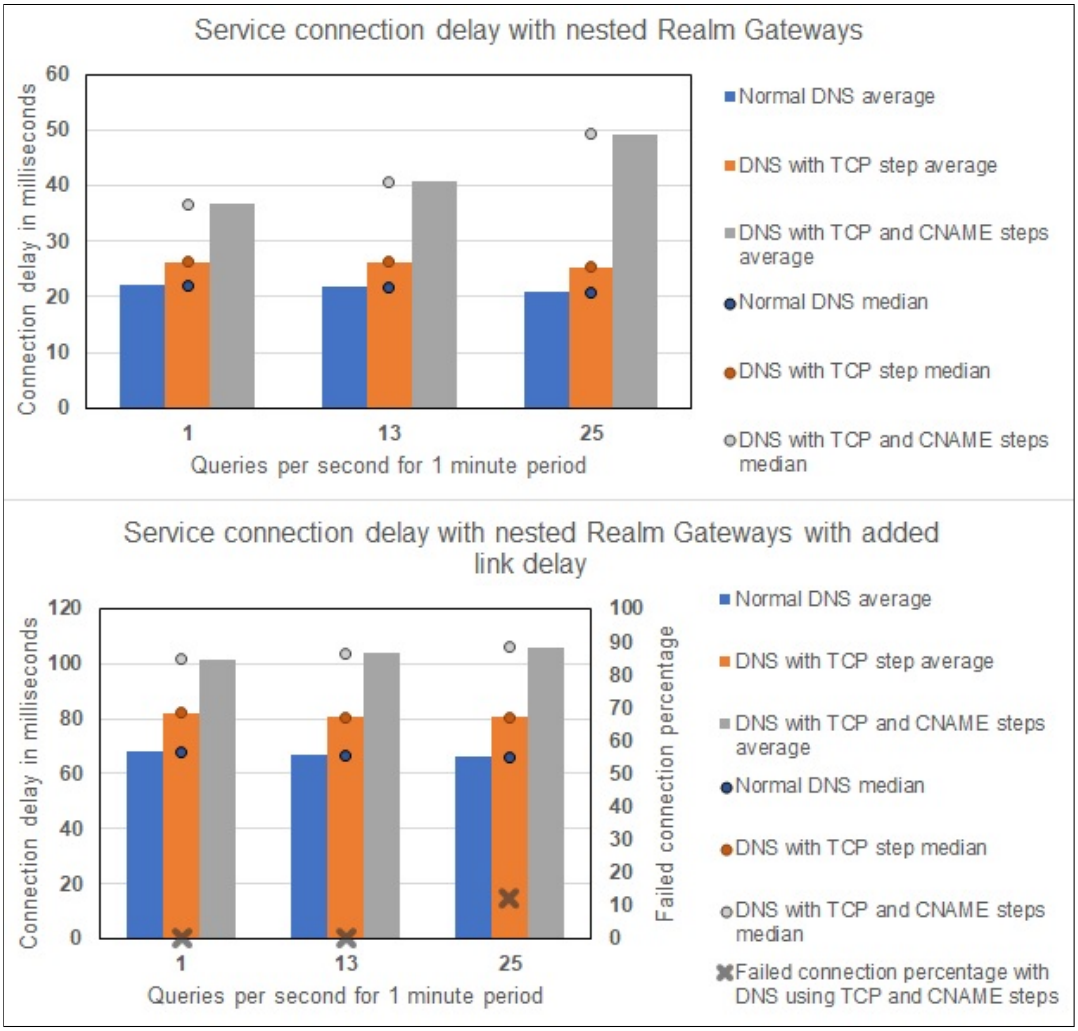

Figure 34: Delay test 4-a and 4-b results for delay for connecting to the protected service with nested Realm Gateway setup

The same was then done for memory use, but now instead of CPU time, the average memory use during the 20 second measuring period was noted. As handling legitimate queries takes a lot of resources, measuring the resource use related to this type of traffic gives a good picture on what level the system operates on higher stress levels.

The resource use measurement 1-a and 1-b results are shown in Figure 35 for CPU usage. One important note here is that the LXCs running the custom DNS and Realm Gateway use CPU time in very limited manner on a base level or when the Realm Gateway or custom DNS software is just running in idle manner or during times of low incoming query amount. With the custom DNS server, the TCP and CNAME usage will add to the CPU load and the same applies to the Realm Gateway. Overall, the Realm Gateway demands more from the CPU, but this isn't surprising as it is far more complicated software. In both cases, the estimated CPU use doesn't go above 10 seconds which could be estimated to mean 50% use of processor resources,

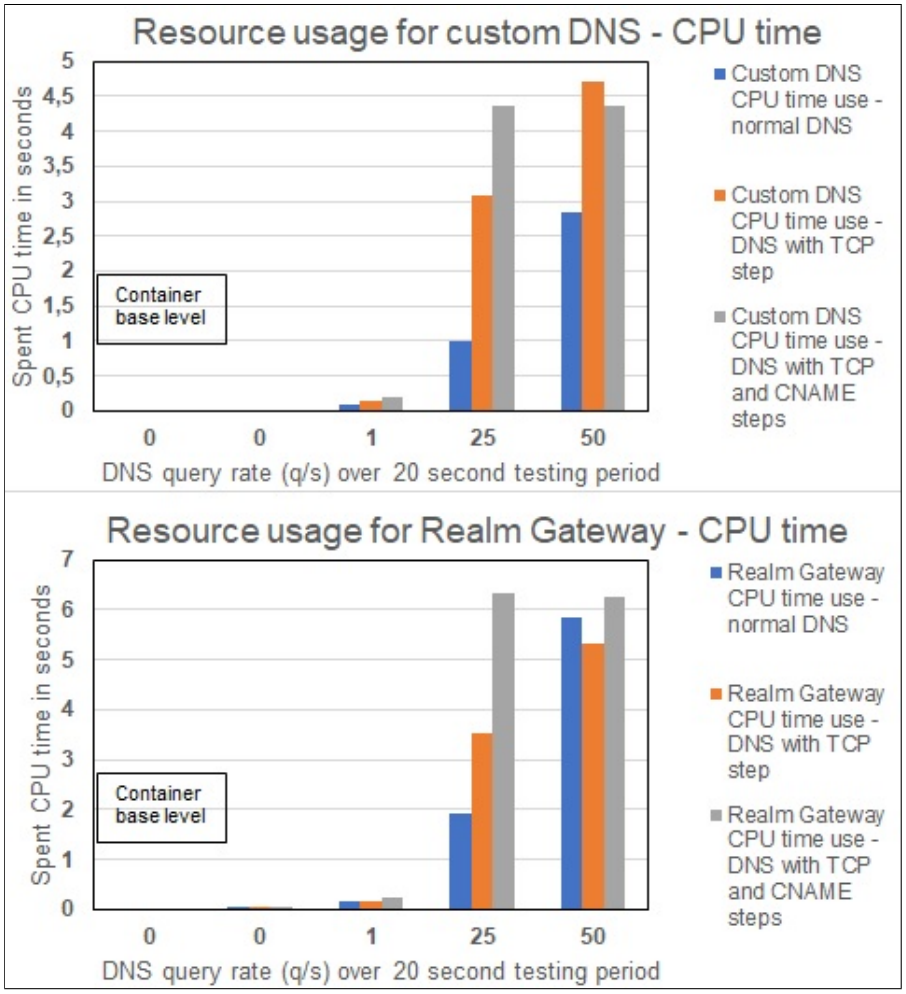

Figure 35: Resource use measurement 1-a and 1-b results

at least for one core. This basically means that there is some room for extra capacity, but it is another matter if and how this unused processor power should be used. For example, if a single software uses up processor resources completely, with modern OSes, it will likely paralyze the system as various OS supporting functions cease to work properly.

Additionally, results for the resource use measurements 2-a and 2-b for memory use are illustrated in Figure 36, again in the upper and lower graph. The memory use for the container's base working level and for the software with both the custom DNS and the Realm Gateway are modest and quite stable, at least if the software is to be run in some modern network node with 1 GB or more of memory. What may affect memory use further is the CNAME information storage, as the Realm Gateway node for example needs to maintain the randomized CNAME query information to compare it to the challenge answer further in the DNS resolution process. There is a periodic memory flush process with the Realm Gateway, however, that will prevent the memory use becoming abnormal in this regard. All in all, with both tested

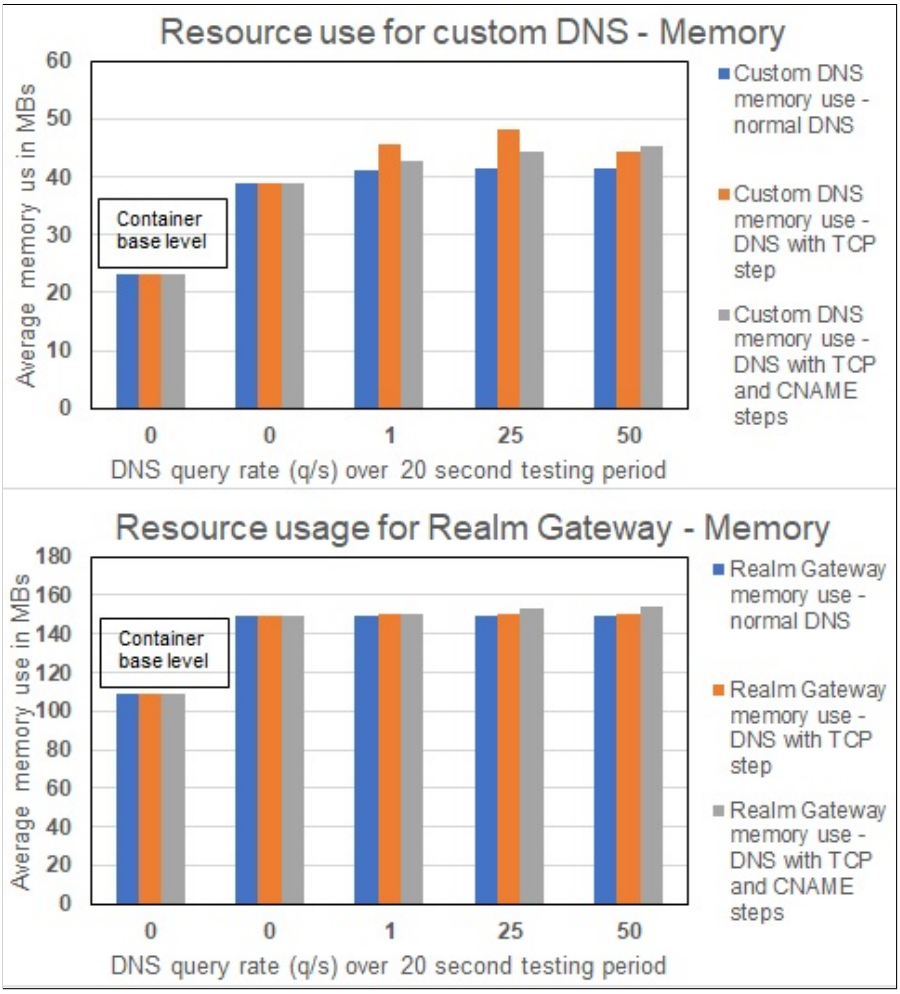

Figure 36: Resource use measurement 2-a and 2-b results

software, the CPU and memory use seemed to be within reasonable limits at least with decently provisioned, modern servers.

5.7 System testing against DoS- and DDoS-attacks

The DoS- and DDoS-attack testing required third testing setup which is shown in Figure 37, mainly for having more computing resources in a private network setting. In this system, all servers were physical machines, depicted in Table 5, connected to a shared physical Ethernet network switch with Ethernet cables. All connected network interfaces supported speeds of 100 Mbit/s and above, and the delays between all the servers and guest LXCs was below 1 millisecond. The desktop PC in this setup hosted the Realm Gateways, protected services and SYN proxies, which obtained addresses from the shared private network by bridging the containers to the desktop's physical network interface. Additionally, one laptop hosted both the public DNS server and custom DNS server on guest containers, again accessed by bridging the laptop's

physical network interface to these containers. Finally, two additional laptops then acted as clients.

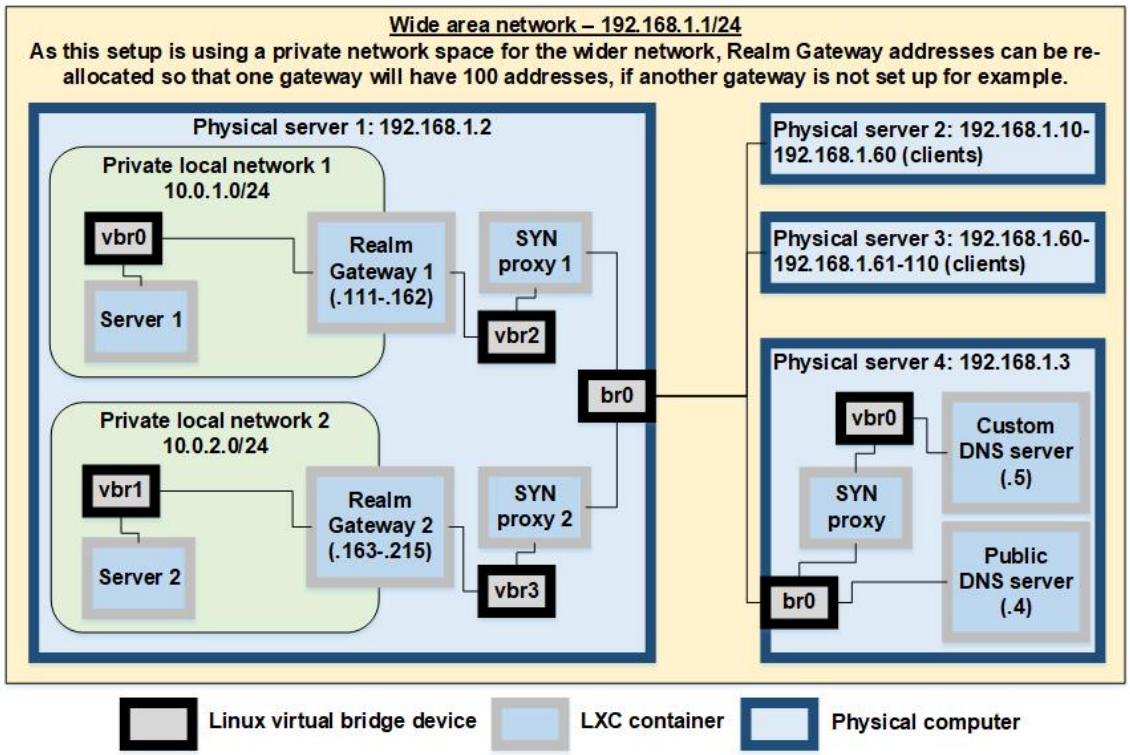

Figure 37: System setup using small private network and physical servers

Majority of the tests presented in this subsection were done in the cloud setting shown in Figure 30, though. The base idea with all tests in this subsection was to generate attack traffic, mostly in the form of problematic DNS queries, towards the custom DNS and Realm Gateway and then measure, in terms of connection delays, how this affects client 1's attempts to connect to the protected service. To reduce the load on the host VM for Realm Gateways, the DoS- and DDoS-tests were done without using the nested gateways. Similarly to earlier delay tests, the protected service was a Python TCP echo server. Generally, all DNS queries were for the Service Fully Qualified Domain Name (SFQDN) on the Realm Gateway using a circular pool of 3 IP addresses. In regard to the network setup, there were no added delays and the actual tests were done as follows:

1. DDoS-test 1 with the cloud setup where system receives varying amount of UDP DNS queries with spoofed source IP addresses and client 1's delay in contacting servers 2 or server 2 and server 4 is to be measured, depending if custom DNS is set up to load balance traffic with randomizing the target Realm Gateway for queries. Test and attack duration is 2 minutes and client tries to contact service every second, with each attempt having 2 re-tries in case of failure with 1 second time-out. The average and median delay of successful

attempts is noted. To elaborate further, here the custom DNS and Realm Gateway have enabled the DNS TCP and CNAME security steps and the two tests compare the case with traffic going against a single Realm Gateway via the custom DNS and the custom DNS randomizing between Realm Gateway 1 and 2.

2. DDoS-test 2 with the cloud setup where a set of 10 attackers would resolve the TCP step successfully during the DNS resolution but then wouldn't respond to the CNAME challenge. Otherwise the test is similar to DDoS-test 1, with the attack query amount being varied and the failed queries also being noted. Bypassing the TCP step is simulated by the custom DNS skipping this step and directing the queries straight to Realm Gateway for CNAME challenge.

3. DDoS-test 3 with the cloud setup where a pool of 65000 source IP addresses is set for the attackers which simulates both a situation where every attack attempt receives a new client reputation or attacker's last contact attempt was so long ago that the reputation has returned to a new client level. Otherwise the test is similar to DDoS-test 2.

4. DDoS-test 4 with the cloud setup where a set of 10 attackers would pass the TCP and CNAME checks and then be allocated an access IP address from the Realm Gateway's circular pool which is then left unused by the attackers. Otherwise the test is similar to DDoS-test 2. Bypassing the TCP and CNAME steps is simulated by the public DNS resolving these steps on behalf of the attacker.

5. DDoS-test 5 with the cloud setup where 10 attackers send DNS queries to a BIND DNS server hosted on the same VM as the public DNS. This DNS server then sends replies directly to the Realm Gateway 1 which simulates a DNS reflection attack. During this time, client 1 tries to connect to server2. In this test, there is no load balancing, but otherwise the results are measured in a similar manner to DDoS-test 2.

6. DoS-test 1 with the cloud setup where 1 attacker sends TCP SYN messages using *hping* software [92] at a varying high rate towards Realm Gateway 1's circular pool addresses while client 1 tries to connect to server 2. The connection delays and failed connection attempts are again measured in a similar manner to DDoS-test 2. In this case, the SYN proxy should prevent the SYN flood from affecting the legitimate client connections.

7. DDoS-test 6 with the small network with physical computers shown in Figure 37. Here the DDoS-attack and result measurement is similar to DDoS-test 4, and instead of comparing the case with 1 Realm Gateway to load balancing with 2, the circular pool address size of the Realm Gateway in question is varied, while also varying the attack query rate. To elaborate further, physical server 2 acts as a legitimate client trying to contact server 1 hosted on the physical server 1 through Realm Gateway 1 with circular address pool of varying size

allocated from the private network address space, while the attack traffic comes from physical server 3. All this traffic naturally goes through the public and custom DNS servers hosted on physical server 4.

5.8 DoS- and DDoS-testing results

With the DoS- and DDoS-attack tests, the attack generally meant that the DNS resolution was done only to a certain step and the attacker would not utilize the allocated circular pool access address if the name resolution came to this point. In order to emulate DDoS-attacks effectively with the limited amount of VMs in the cloud, the public DNS server and the custom DNS server were able to turn off the TCP security steps and/or the CNAME steps. This would make it possible for the DNS system to emulate attacks that would appear to come from multiple semi-legitimate sources that would appear to pass the TCP and CNAME steps from the Realm Gateway's point of view. This was also facilitated by public DNS forwarding information about the client's IP address with DNS ECS, which would then be forwarded to the Realm Gateway by the custom DNS. As a reminder, the delay in the tests here was measured for legitimate connections from sending the initial DNS query to contacting the protected TCP service, doing 1 query/contact attempt per second during the 2-minute testing periods with 2 possible re-tries for DNS name resolution.

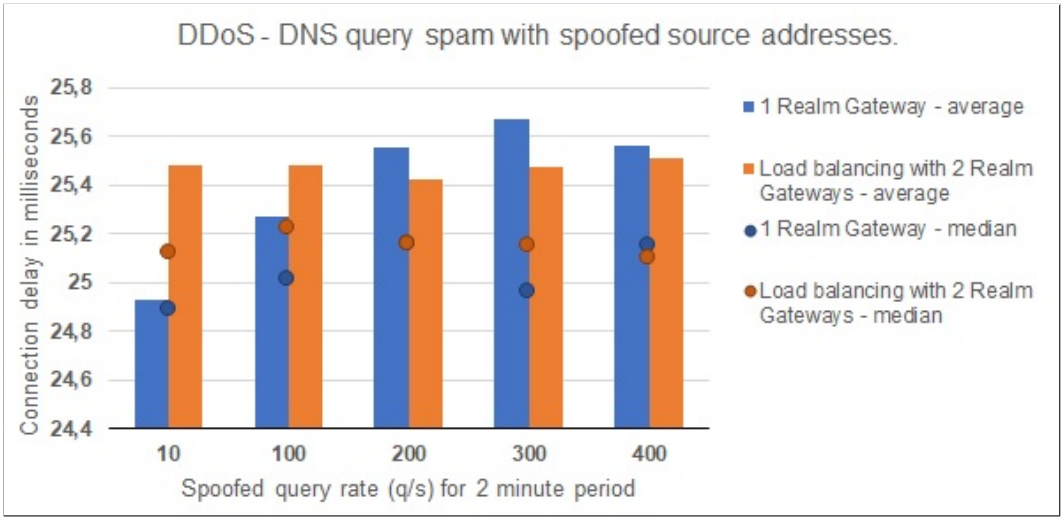

Figure 38: DDoS-test 1 results where UDP DNS flood was used

The results for DDoS-test 1 are shown in Figure 38. The custom DNS could easily handle attack rates up to 400 queries per second which is the combined iptables rate limit of the 2 Realm Gateways for DNS queries. It seems that the TCP affirmation step is a good way to deter basic DNS UDP floods at least on an application level, and the spoofed queries didn't even reach the Realm Gateway. Effect on legitimate traffic was negligible with the delay average differences being within 1 millisecond. It

is another matter what happens if the attack rate goes far above the possible iptables UDP DNS limits of the custom DNS and the Realm Gateway. In that case, high percentage of legitimate traffic could be dropped by the firewall, depending on the scale of the attack. Load balancing in this test didn't have any noticeable effect on one way or another.

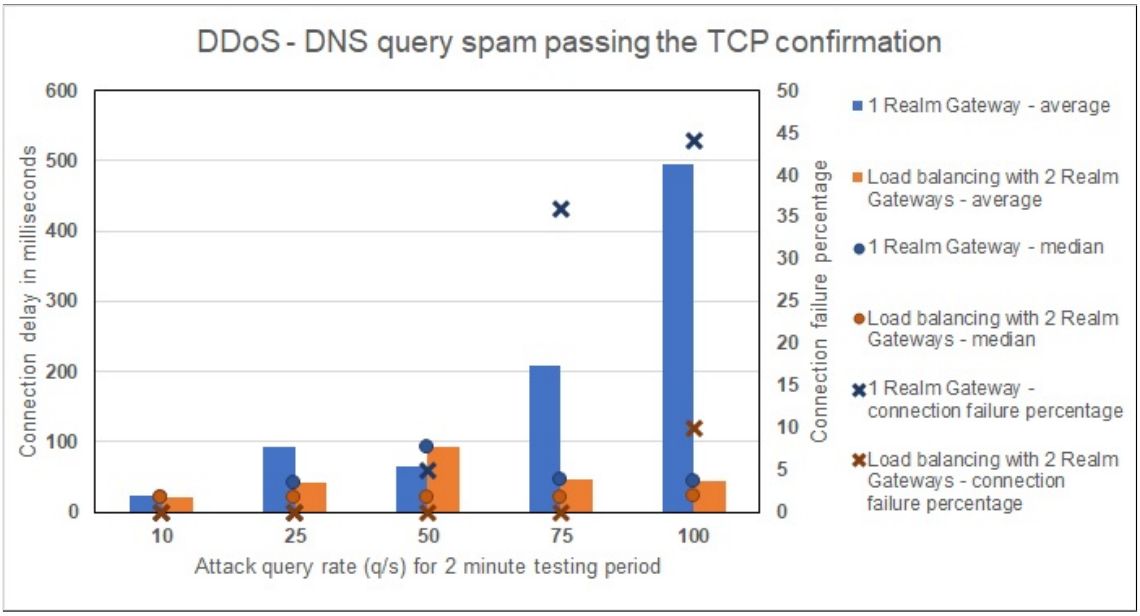

Figure 39: DDoS-test 2 results where DNS TCP step was bypassed

In regard to more advanced DDoS-attack, the results for DDoS-test 2 are presented in Figure 39. In this test, the attackers appeared to pass the TCP step and didn't answer to the CNAME challenge, while appearing to become from 10 different subnets, separate from the legitimate client. Here the simulated attack rate scale didn't go beyond 100 queries per second, as higher rates caused too many failed connection attempts for the valid client. Smaller scale may not be as critical fault here as one could quickly surmise as mounting large amount of attack nodes that are able to finish TCP communication setups is more challenging than just spamming the network with bogus UDP traffic. Here 1 Realm Gateway could manage attack traffic reasonably well up to 50 queries per second. After this point, the average delay and failed connection attempts would increase a lot, although successful connections could still be done within reasonable time-frame, generally under 500 milliseconds.

Load balancing seemed to help a lot with DDoS-test 2, making the connection process work reasonably well even with 100 attack queries per second. Main reason for this seemed to be the CNAME memory flushing procedure's temporary, slowing effect on the Realm Gateway performance. With 2 Realm Gateways working simultaneously, they would not necessarily flush memory at a same time, which would induce less stress on the whole system during the flushing procedures. Additionally, the reputation system seemed to contain the 10 attackers reasonably fast, but even if their

replies were not handled fully, there was still an ongoing negative effect present on the system performance.

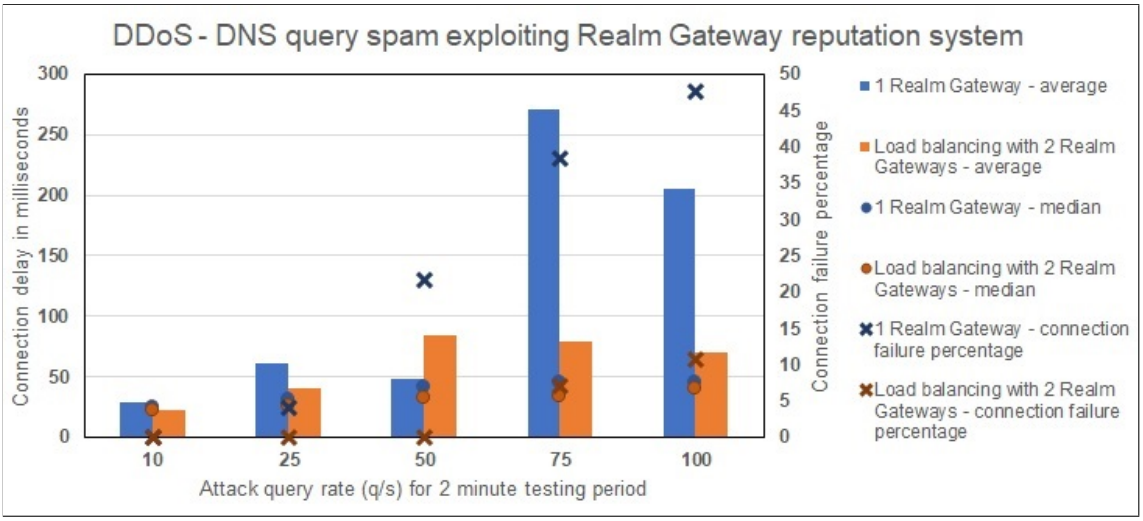

Figure 40: DDoS-test 3 results where DNS TCP step was bypassed

The results for DDoS-test 3 are shown in Figure 40. With this test, the attacking clients would appear to come from an unseen subnet with high probability. This emulated the case where every incoming attack query would appear to be a first query for the client or a query from a client whose reputation was reset back to new client levels after a period of inactivity. The results mirror here DDoS-test 2 as the single Realm Gateway gets into trouble when the attack rate goes above 50 queries per second. Again, successful queries are done within reasonable time and load balancing helps, especially with avoiding failed connections. At least based on this test, the Realm Gateway seems to use similar amount of resources handling new clients than handling clients with poor reputation, which may be a problem in the system design.

In regard to even more advanced DDoS-attack, the results for DDoS-test 4 are illustrated in Figure 41. Here the attacker would pass both the TCP and CNAME steps from the Realm Gateway's point of view with the public DNS behavior emulation. It seems that this type of attack is a really bad problem for the Realm Gateway as the system suffers larger delays and more importantly most of the connection attempts for DNS resolution fail when the attack traffic rate goes over the legitimate client query rate. Unfortunately, load balancing doesn't help much here. The root of the problem is that the allocation process from the Realm Gateway's circular pool does not scale, if the pool size is limited, as here the size was just 3 addresses.

For DDoS-test 5, the results are shown in Figure 42, where the attack traffic here goes directly towards the Realm Gateway, as the attack would involve oblivious DNS servers who might have direct access to the Realm Gateway. In a similar manner to

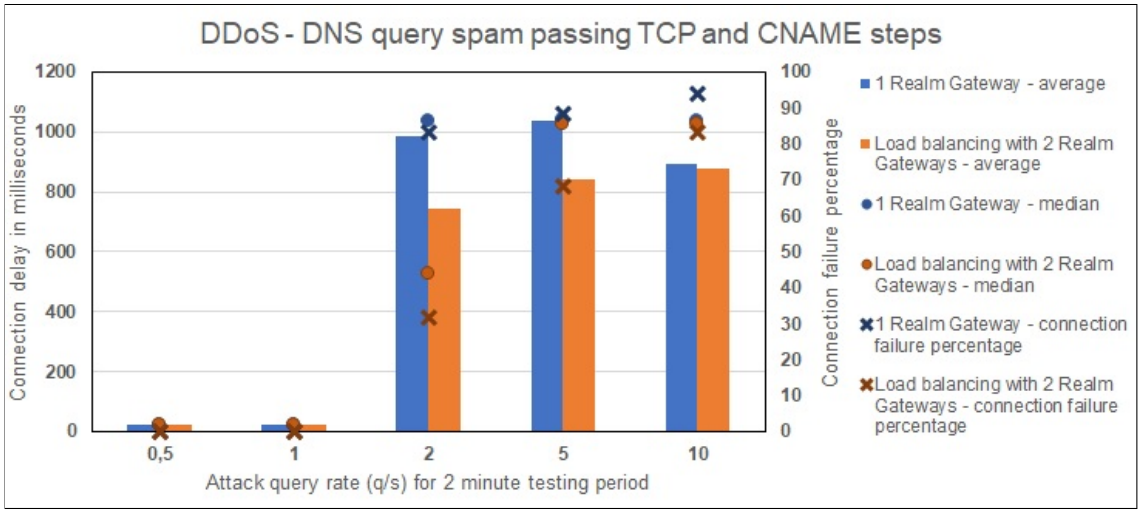

Figure 41: DDoS-test 4 results where DNS TCP and CNAME steps were bypassed

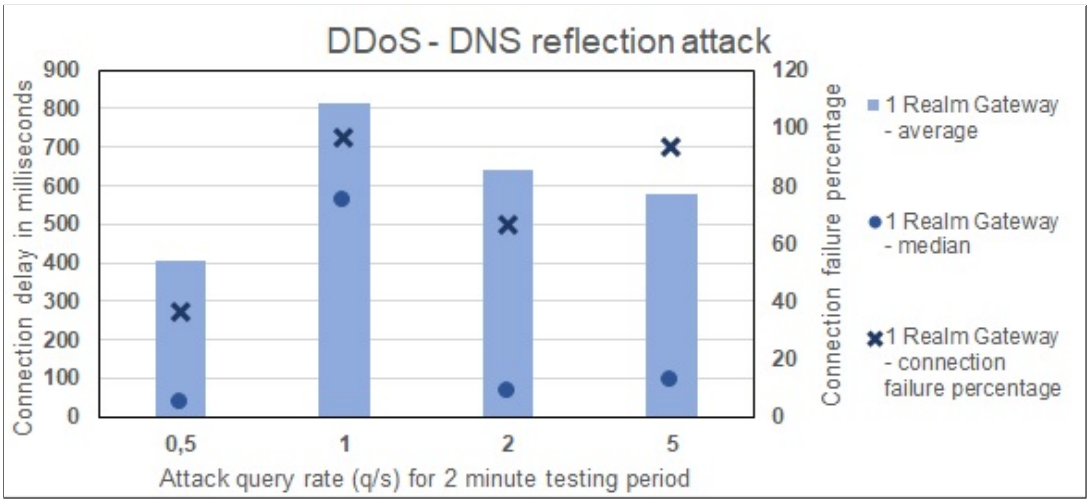

Figure 42: DDoS-test 5 results where DNS reflection attack was utilized

the previous test, the Realm Gateway gets into big trouble with very limited attack rate. In contrast to the address allocation issues, here the Realm Gateway seems to use far too many system resources to deal with erroneous DNS queries. This implies that the Realm Gateway robustness should definitely be improved versus non-standard DNS queries compared to the ”standard” case of Realm Gateway just receiving neat, recursive DNS queries that always contain the client subnet address data. At least in principle, clearly faulty DNS queries such as responses to non-existent queries, as is the case with DNS reflection attacks, are relatively easy to filter out.

The last test with the cloud setup was DoS-test 1, or the SYN flood test with just 1 attacker sending TCP SYN packets towards Realm Gateway’s circular pool addresses with the SYN proxy being the defense measure. The results of this test are shown

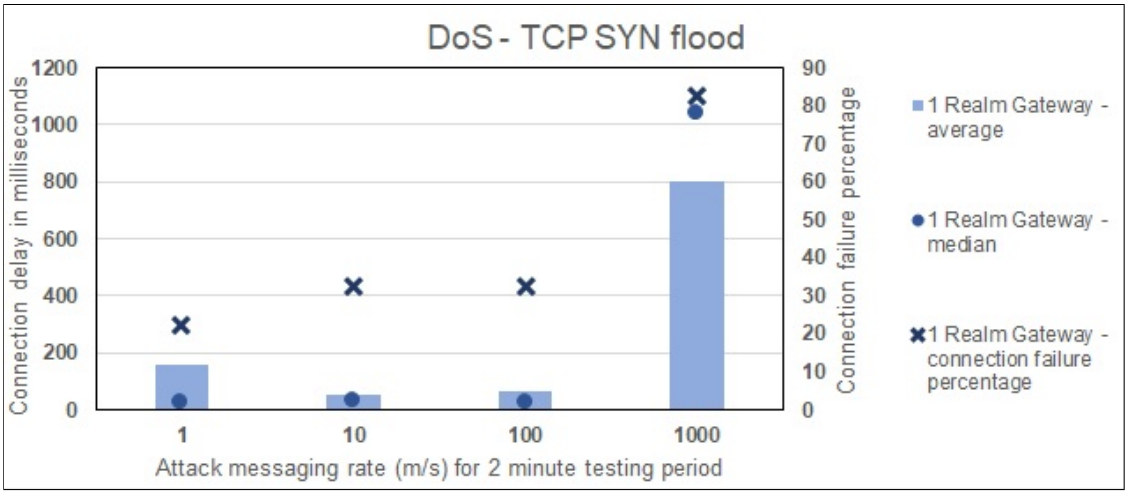

Figure 43: DoS-test 1 results where SYN flood was utilized

in Figure 43. For some reason, the SYN proxy script provided with the Realm Gateway incurred lots of failed connection attempts for the legitimate users. At least with higher attack rates, this makes it very problematic to utilize the proxy script effectively as it hampers legitimate clients too much. It is possible that the SYN proxy in the test was perhaps configured poorly or it didn't fully work with the given containers and VMs in the cloud, as there were many virtual network interfaces present. This means that it is wise to do further testing with the given SYN proxy with adjusted configurations in different network setups or to choose a different proxy implementation altogether, as it is an important part of the Realm Gateway system to prevent TCP flood towards the DNS system TCP servers and towards the Realm Gateway's NAT access addresses.

Finally, the results for DDoS-test 6, done with the small network setup with physical servers, is shown in Figure 44. This test is linked to the DDoS-test 4, where it is observed here, if increasing the circular pool size would help with DDoS-attacks which are able to finish up the DNS name resolution, passing the TCP and CNAME checks and then not claim the allocated access address. It seems that by just increasing the circular pool to size 10 will eliminate the connection issues that were the problem with DDoS-test 4. This would imply that Realm Gateways should be allocated large address pools during the network and system setup. This may not be possible with public IPv4 addresses especially, so another option to help with this kind of DDoS-attack would be to enhance the address allocation algorithms connected to successful DNS name resolution. It is worth mentioning that there is an existing alternative for Realm Gateway's circular pool allocation algorithm, which is the airwall system that has been developed for the Aalto 5G project. This system could be linked fully to the Realm Gateway in future as it circumvents the address allocation limitations of the current Realm Gateway implementation.[141]

As a concluding note, there needs to be some improvements made for the Realm

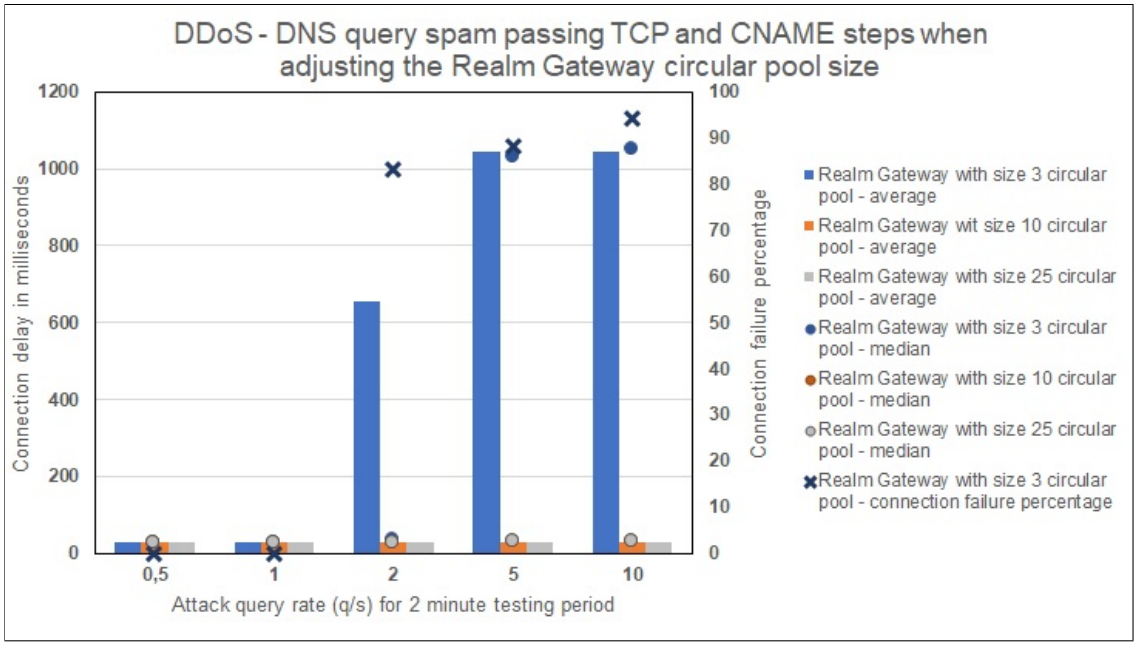

Figure 44: DDoS-test 6 results where DNS TCP and CNAME steps were bypassed

Gateway DNS message handling process and circular pool address allocation specifically. In regard to question if load balancing is necessary, having multiple Realm Gateways in place helps in some scenarios, but to get the most out of them would require signaling between the gateways to synchronize memory management.

6 Conclusions

It is apparent that network service security is a very complex problem, where there seems to be a constant struggle between the attackers and the defenders, as new software and hardware exploits are found out and new attack methods are designed, and then the defending side scrambles to find effective security measures in response to these new threats. If web services utilize up-to-date software including access control methods and communication encryption, the most apparent threat is indeed various Denial-of-Service attacks which are usually distributed to come from many network nodes.

In regard to the research problem, several tasks were accomplished. First, the validation tests with the Realm Gateway and the custom DNS module did show that it is possible to detach critical services to private network space for additional security, where they could still be accessed from the public network space using common network protocols which makes the Realm Gateway and accompanied modules practical at least to some degree. This is supported by the resource use measurements which did show that Realm Gateway and the custom DNS use up reasonable amount of system resources. Second, the Realm Gateway in conjunction with the custom DNS did perform well against some DDoS-attacks which points it being a useful security defense measure at least against certain threats that target web service availability.

At this point is good to note that no matter what kind of contained security software is used, there may be no singular solution against all kinds of DoS- and DDoS-attacks. All-encompassing defenses could require complex, networked systems working on multiple layers in the TCP/IP model while also maintaining signaling schemes for communication between components in the defensive system, as was showcased with the state-of-the-art DoS- and DDoS-containment methods in Chapter 3. The Realm Gateway by Jesús Llorente Santos can however be a good basis for secure web services, especially as the Realm Gateway development is an ongoing process. This is collaborated by earlier research in addition to this thesis, where the Realm Gateway and related system principles have been successfully tested in detaching and protecting web services and defending against certain types of DoS-attacks.[122][125]

Even if the basic idea of the Realm Gateway is sound, there is room for improvement, as is the case with most complex software. Various issues that have come about during the writing of this thesis are listed below and should be taken into consideration when further Realm Gateway improvements are being designed:

- For one, the Realm Gateway seems to be still a work-in-progress software, so there is a need to increase the robustness of the circular pool address allocation procedure and the handling of problematic or erroneous DNS queries. Utilizing the aforementioned airwall system alternative could help with this.[141]
- In a similar vein, it would be important to make Realm Gateway's DNS resolution process more resistant to reasonable network delays, as it can't

be assumed that every node in the system is within the delay range of 10 milliseconds or less. Code improvements and using alternative programming language for the Realm Gateway and custom DNS implementation may help with these issues as they connect to the limitations of Python performance which is discussed in more detail in Appendix C.

- It is practical to set up the Realm Gateway system itself, but another question is the co-operation of nearby DNS services. In order for the address allocation to function, Realm Gateway relies on the DNS ECS subnet information to identify incoming clients, so the DNS servers doing recursive queries just before the custom DNS relay should be persuaded to send the ECS information forward. Additionally, these DNS servers would need specific configurations to direct DNS replies with the truncated flag up and DNS CNAME replies back to the actual client instead of doing these steps automatically themselves. These are not impossible tasks, but they would likely require the Realm Gateway maintainer to purchase these services from the DNS maintainers or to get direct control over the nearby DNS systems himself.

- It would be good to ensure the proper working of the SYN proxy component, as the testing in this thesis implied that the negative effect for legitimate clients when SYN flood interfered with Realm Gateway's access addresses was too drastic. As was discussed in Chapter 6, this may have just come from SYN proxy incompatibility with the virtual machines and network components on the cloud setup or from SYN proxy configuration errors.

- It would be prudent to support DNSSEC, as without DNS security measures, MitM-attacks are possible, where attackers could forge replies coming from the Realm Gateway or from the custom DNS.

- If custom DNS is to be used, it would likely be smart to establish permanent, secure TCP connections between the Realm Gateway and respective custom DNS node for DNS messaging to make it more difficult for attackers to spam Realm Gateway with problematic DNS messages. This also involves setting up proper firewall filtering rules from the get-go to custom DNS server and Realm Gateway to weed out traffic not related to DNS.

In additional positive note, Realm Gateway brings about very useful ideas for application level defenses against DoS- and DDoS-attacks, and many benefits may come from simple fine-tuning of the software implementation. As was also noted in the previous chapter, having only limited capacity for legitimate clients may not be a problem, as the purpose of the Realm Gateway system would be to protect critical services which have limited user base and would not likely suffer sudden spikes in incoming legitimate user connections. In most cases, the legitimate client amounts can be easily predicted, and the service would then be provisioned accordingly with additional capacity being reserved to handle DoS- and DDoS-attacks.

It is also important to make a disclaimer that only limited conclusions can be drawn from the performance tests done in this thesis with the Realm Gateway and with the custom DNS relay. As testing was done in virtual environments, which introduces processing overhead, one could get a better view of the system performance and likely improved results, if additional tests would be done running the Realm Gateway software on dedicated servers without virtualization. Some useful information can be gleaned from the test results, however, especially when comparing the different systems used in the thesis. For one, deploying nested Realm Gateways will introduce additional, notable delays for contacting the protected web service, and it is questionable if this is worth it, even if it brings about enhanced system security. For another, the load balancing scheme improved system performance to some degree, but to make better use of it, one likely has to develop signaling between the Realm Gateways and the custom DNS relay, so that data traffic can be directed away from nodes that are doing system maintenance operations for example. This could mean that, at least with the current Realm Gateway code, it is not clear if load balancing should be used on all deployments due to the added system complexity.

6.1 Future work

First, it has to be understood that the tests done with this thesis were certainly not all-encompassing. Therefore, additional tests should be done with DoS- and DDoS-traffic and general Realm Gateway and custom DNS performance with more realistic network settings, where each network node would have at least some network delay, as this would be the case when the gateway would protect actual web service. Additionally, as was implied just before, the Realm Gateway and custom DNS behavior should be tested in non-virtualized environments, with network nodes that have two or more physical network interfaces to compare performance between those and virtual network interfaces.

As a second point, one major future improvement would be creating a signaling system between the Realm Gateways and the custom DNS relays. With the signaling, Realm Gateways could exchange client reputation information in order to spread black-listed client identities, so they could be quickly blocked at other Realm Gateways if necessary. Another clear benefit from this type of signaling would be synchronizing memory management between multiple Realm Gateways in a load balance setting, where this information could also be passed to the custom DNS in order for it to reduce traffic temporarily to a gateway that is currently flushing CNAME data from memory.

The signaling could be then extended to cover different upstream nodes such as high-capacity routers, where there is a possibility to do effective, large-scale rate-limiting. This would help against major flooding attacks, that would overwhelm Realm Gateway or custom DNS links and the respective, local rate-limits as problematic traffic would be limited before it even reaches the Realm Gateway system. This would not be simple and cheap addition, so it remains to be seen how feasible this type of

solution would be.

In regard to practical use of the system, it would be beneficial to have more approachable real-time system controls in place, so that the Realm Gateway functions could be adjusted on the fly, while also having concise documentation available on the Realm Gateway functions and procedures. This also links to improving the Realm Gateway DNS name resolution process against aberrant DNS queries as was discussed earlier. It would also be wise to explore if the BGP ECMP routing design could replace the custom DNS relays when load balancing is considered, if adding to BGP systems would be more straightforward than adding additional components to the DNS system,

Finally, for improved system security, the Realm Gateway could expand the reputation system and traffic monitoring to pass-through traffic. Currently, if the DNS name resolution process succeeds and the client claims the access address, the Realm Gateway doesn't really care what kind of traffic goes through if it passes the SYN proxy. This may be problematic in the case of non-DNS application level DoS-attacks, so at least a framework of noting or filtering out certain kinds of high-level traffic such as abnormal rates of HTTP request for the same resource could be useful. Obviously, this would incur even more processing overhead and it may be difficult to say what kind of messaging exactly is to be noted or affected. There are also legal limitations on what level the ongoing communication can be monitored or manipulated in a public network.

As a concluding note, similarly to most software development cases, the Realm Gateway, the accompanied SYN proxy and the custom DNS likely have bugs and general limitations that could be ironed out and removed with diligent updating. It can also be said that the Realm Gateway is an interesting and useful network security concept, so there is a lot of potential in using it directly or getting good ideas from it to other security system implementations.

References

[1] M. A. Bishop. 2005. *Introduction to computer security.* Addison-Wesley. pp. 1–22.

[2] L. L. Knud. 2018. *State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating.* Web document. Retrieved 27.9.2018. Available: https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/.

[3] D. Medhi and K. Ramasamy. 2007. *Network Routing : Algorithms, Protocols, and Architectures.* Morgan Kaufmann. pp. 1–27.

[4] J. Postel. 1981. *Internet Protocol.* RFC 791.

[5] S. Deering and R. Hinden. 2017. *Internet Protocol, Version 6 (IPv6) Specification.* RFC 8200.

[6] Javvin. 2005. *Network Protocols Handbook,* 2nd ed. Javvin Technologies Inc. p. 27.

[7] C. Osborne. 2017. *The average DDoS attack cost for businesses rises to over $2.5 million.* Web document. Retrieved 21.5.2018. Available: https://www.zdnet.com/article/the-average-ddos-attack-cost-for-businesses-rises-to-over-2-5m/.

[8] C. Kolias, et al. 2017. DDoS in the IoT: Mirai and Other Botnets. In: *IEEE Computer,* vol. 50, no. 7, pp. 80-–84.

[9] YLE. 2018. *Cyber attack shuts down many government websites in Finland.* Web document. Retrieved 26.10.2018. Available: https://yle.fi/uutiset/osasto/news/cyber_attack_shuts_down_many_government_websites_in_finland/10350316.

[10] P. Ferguson. 2000. *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing.* RFC 2827.

[11] A. Leon-Garcia and I. Widjaja. 2004. *Communication networks,* 2nd ed. McGraw-Hill. pp. 34–92.

[12] P. Mockapetris. 1987. *Domain Names – Concepts and Facilities.* RFC 1034.

[13] S. Hares, T. Li and Y. Rekhter. 2006. *A Border Gateway Protocol 4 (BGP-4).* RFC 4271.

[14] R. Kantola, H. Kabir and P. Loiseau. 2017. Cooperation and end-to-end in the Internet. In: *International Journal of Communication Systems,* vol. 30, no. 12, e3268.

[15] J. L. Santos. 2018. *Realm Gateway.* Software. Available: https://github.com/Aalto5G/RealmGateway.

[16] J. Postel. 1981. *Transmission Control Protocol.* RFC 793.

[17] T. Berners-Lee, et al. 1999. *Hypertext Transfer Protocol – HTTP/1.1.* RFC 2616.

[18] R. Bennett. 2009. *Designed for Change: End-to-End Arguments, Internet Innovation, and the Net Neutrality Debate.* Web Document. Retrieved 27.10.2018. Available: https://www.itif.org/files/2009-designed-for-change.pdf.

[19] R. Braden. 1989. *Requirements for Internet Hosts – Communication Layers.* RFC 1122.

[20] J. Postel. 1980. *User Datagram Protocol.* RFC 768.

[21] D. C. Plummer. 1982. *An Ethernet Address Resolution Protocol.* RFC 826.

[22] E. Rescorla. 2018. *The Transport Layer Security (TLS) Protocol Version 1.3.* RFC 8446.

[23] V. Ramachandran and S. Nandi. 2005. Detecting ARP spoofing: An active technique. In: *International Conference on Information Systems Security,* Springer, pp. 239–250.

[24] E. Rescorla. 2000. *HTTP Over TLS.* RFC 2818.

[25] Y. Rekhter. 1993. *An Architecture for IP Address Allocation with CIDR.* RFC 1518.

[26] M. Cotton and L. Vegoda. 2013. *Special-Purpose IP Address Registries.* RFC 6890

[27] J. Postel. 1981. *Internet Control Message Protocol.* RFC 792

[28] Internet Society. 2018. *State of IPv6 Deployment 2018.* Web Document. Retrieved 28.10.2018. Available: https://www.internetsociety.org/resources/2018/state-of-ipv6-deployment-2018/.

[29] Internet Assigned Numbers Authority. 2018. *Service Name and Transport Protocol Port Number Registry.* Web Document. Retrieved 29.10.2018. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?&page=1.

[30] N. Brownlee, C. Mills and G. Ruth. 1999. *Traffic Flow Measurement: Architecture.* RFC 2722.

[31] J. Mirkovic and P. Reiher. 2004. A taxonomy of DDoS attack and DDoS defense mechanisms. In: *ACM SIGCOMM Computer Communication Review,* vol. 34, no. 2, pp. 39–53.

[32] P. Watson. 2004. *Slipping in the Window: TCP Reset attacks.* Web Document. Retrieved 30.10.2018. Available: http://67.225.133.110/~gbpprorg/2600/SlippingInTheWindow_v1.0.pdf.

[33] G. Malkin. 1998. *RIP Version 2.* RFC 2453.

[34] J. Moy. 1998. *OSPF Version 2.* RFC 2328.

[35] R. Atkinson. 2007. *RIPv2 Cryptographic Authentication.* RFC 4822.

[36] E. Jones and O. Le Moigne. 2006. *OSPF Security Vulnerabilities Analysis.* Web Document. Retrieved 4.11.2018. Available: https://tools.ietf.org/html/draft-ietf-rpsec-ospf-vuln-02.

[37] Q. Vohra and E. Chen. 2012. *BGP Support for Four-Octet Autonomous System (AS) Number Space.* RFC 6793.

[38] S. Sangli and D. Tappan. 2006. *BGP Extended Communities Attribute.* RFC 4360.

[39] S. Snijders, J. Heasley and M. Schmidt. 2017. *Use of BGP Large Communities.* RFC 8195.

[40] R. Chandra, P. Traina and T. Li. 1996. *BGP Communities Attribute.* RFC 1997.

[41] M. G. Marsh. 2001. *Policy Routing With Linux - Online Edition.* Web Document. Retrieved 4.11.2018. Available: http://www.policyrouting.org/PolicyRoutingBook/ONLINE/TOC.html.

[42] K. Ishiguro, et al. 2005. *Quagga 1.2.0.* Software. Available: https://www.quagga.net/.

[43] M. Lepinski and K. Sriram. 2017. *BGPsec Protocol Specification.* RFC 8205.

[44] C. Alaettinoglu. 2015. *BGP Security: No Quick Fix.* Web Document. Retrieved 6.11.2018. Available: https://www.networkcomputing.com/networking/bgp-security-no-quick-fix/1303232068.

[45] bgphelp.com. 2017. *2017 BGP Table Size Prediction and Potential Impact on Stability of Global Internet Infrastructure.* Web Document. Retrieved 6.11.2018. Available: http://bgphelp.com/2017/01/01/bgpsize/.

[46] P. Stavroulakis and M. Stamp, editors. 2010. *Handbook of information and communication security.* Springer Science & Business Media. pp. 177–219.

[47] P. Srisuresh and M. Holdrege. 1999. *IP Network Address Translator (NAT) Terminology and Considerations.* RFC 2663.

[48] F. Audet and C. Jennings. 2007. *Network Address Translation (NAT) Behavioral Requirements for Unicast UDP.* RFC 4787.

[49] J. Rosenberg, et al. 2002. *SIP: Session Initiation Protocol.* RFC 3261.

[50] Teluu Inc. 2009. *Introduction to Network Address Translation (NAT) and NAT Traversal.* Web Document. Retrieved 6.11.2018. Available: https://www.pjsip.org/pjnath/docs/html/group__nat__intro.htm.

[51] Y. Takeda. 2003. *Symmetric NAT Traversal using STUN.* Web Document. Retrieved 6.11.2018. Available: https://tools.ietf.org/id/draft-takeda-symmetric-nat-traversal-00.txt.

[52] R. Mahy, P. Matthews and J. Rosenberg. 2010. *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN).* RFC 5766.

[53] A. Keranen, C. Holmberg and J. Rosenberg. 2018. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal.* RFC 8445.

[54] Ubuntu Documentation. 2011. *Servers Behind NAT.* Web Document. Retrieved 6.11.2018. Available: https://help.ubuntu.com/community/ServersBehindNAT.

[55] F5 Networks Inc. 2016. *The Myth of Network Address Translation as Security.* Web Document. Retrieved 6.11.2018. Available: https://www.f5.com/services/resources/white-papers/the-myth-of-network-address-translation-as-security.

[56] P. Albitz and C. Liu. 2006. *DNS and Bind,* 5th ed. O'Reilly.

[57] P. Mockapetris. 1987. *Domain Names – Implementation and Specification.* RFC 1035.

[58] S. Pillai. 2013. *difference between iterative and recursive dns query.* Web Document. Retrieved 6.11.2018. Available: https://www.slashroot.in/difference-between-iterative-and-recursive-dns-query.

[59] ICANN. 2018. *ICANN DNS Engineering.* Web Document. Retrieved 6.11.2018. Available: https://www.dns.icann.org/.

[60] J. Damas, M. Graff and P. Vixie. 2013. *Extension Mechanisms for DNS (EDNS(0)).* RFC 6891.

[61] P. Vixie. 1996. *A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY).* RFC 1996.

[62] P. Vixie, et al. 1997. *Dynamic Updates in the Domain Name System (DNS UPDATE).* RFC 2136.

[63] C. M. Kozierok. 2017. *TCP/IP Guide.* Web Document. Retrieved 11.11.2018. Available: http://www.tcpipguide.com/index.htm.

[64] Internet Assigned Numbers Authority. 2018. *Domain Name System (DNS) Parameters.* Web Document. Retrieved 11.11.2018. Available: https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml.

[65] Internet Systems Consortium. 2018. *BIND.* Software. Available: https://www.isc.org/downloads/bind/.

[66] Cisco. 2018. *DNS Resource Records.* Web Document. Retrieved 11.11.2018. Available: https://www.cisco.com/c/en/us/support/docs/ip/domain-name-system-dns/12684-dns-resource.html.

[67] dnssec.net. 2018. *DNSSEC: DNS Security Extensions Securing the Domain Name System.* Web Document. Retrieved 11.11.2018. Available: https://www.dnssec.net/.

[68] R. Arends, et al. 2005. *DNS Security Introduction and Requirements.* RFC 4033.

[69] C. Contavalli, et al. 2016. *Client Subnet in DNS Queries.* RFC 7871.

[70] D. Atkins and R. Austein. 2004. *Threat Analysis of the Domain Name System (DNS).* RFC 3833.

[71] B. Woodcock. 2002. *Best Practices in IPv4 Anycast Routing.* Web Document. Retrieved 11.11.2018. Available: https://www.pch.net/resources/Papers/ipv4-anycast/ipv4-anycast.pdf.

[72] R. Arends, et al. 2005. *Resource Records for the DNS Security Extensions.* RFC 4034.

[73] R. Arends, et al. 2005. *Protocol Modifications for the DNS Security Extensions.* RFC 4035.

[74] Microsoft Windows Dev Center. 2018. *Public Key Infrastructure.* Web Document. Retrieved 11.11.2018. Available: https://docs.microsoft.com/en-us/windows/desktop/SecCertEnroll/public-key-infrastructure.

[75] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone. 1996. *Handbook of Applied Cryptography.* CRC Press. pp. 223–271.

[76] J. Daemen and V. Rijmen. 2007. *AES Proposal: Rijndael.* Web Document. Retrieved 14.11.2018. Available: https://web.archive.org/web/20070203204845/https://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf.

[77] Cryptography World. 2004. *The Cryptography Guide: Triple DES.* Web Document. Retrieved 14.11.2018. Available: https://web.archive.org/web/20170312125442/http://www.cryptographyworld.com/des.htm.

[78] W. Simpson. 1996. *PPP Challenge Handshake Authentication Protocol (CHAP).* Web Document. Retrieved 15.11.2018. Available: http://tools.ietf.org/html/rfc1994.

[79] E. W. Weisstein. 2018. *Wolfram MathWorld - Number Field Sieve.* Web Document. Retrieved 19.11.2018. http://mathworld.wolfram.com/NumberFieldSieve.html.

[80] R. Impagliazzo and R. Paturi. 2001. On the complexity of k-SAT. In: *Journal of Computer and System Sciences,* vol. 62, no. 2, pp. 367-–375.

[81] T. H. Cormen, et al. 2009. *Introduction to algorithms,* 3rd ed. MIT press. pp. 43–64.

[82] T. Kleinjung, et al. 2010. Factorization of a 768-bit RSA modulus. In: *Annual Cryptology Conference,* Springer, pp. 333–350.

[83] C. R. Dougherty. 2009. *MD5 vulnerable to collision attacks.* Web Document. Retrieved 22.11.2018. Available: https://www.kb.cert.org/vuls/id/836068/.

[84] Y. Sheffer, R. Holz and P. Saint-Andre. 2015. *Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS).* RFC 7457.

[85] J. Leyden. 2013. *Gmail, Outlook.com and e-voting 'pwned' on stage in crypto-dodge hack.* Web Document. Retrieved 22.11.2018. Available: https://www.theregister.co.uk/2013/08/01/gmail_hotmail_hijacking/.

[86] E. Eastlake 3rd and T. Hansen. 2011. *US Secure Hash Algorithms (SHA and SHA-based HMAC and HK.* RFC 6234.

[87] S. T. Zargar, J. Joshi and D. Tipper. 2013. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. In: *IEEE communications surveys & tutorials,* vol. 15, no. 4, pp. 2046–2069.

[88] B. Posey. 2006. *Networking Basics: Part 1 - Networking Hardware.* Web Document. Retrieved 30.11.2018. Available: http://techgenix.com/networking-basics-part1/.

[89] Microsoft. 2018. *Network data buffer management.* Web Document. Retrieved 30.11.2018. Available: https://docs.microsoft.com/en-us/windows-hardware/drivers/netcx/network-data-buffer-management.

[90] A. Hussain, J. Heidemann and C. Papadopoulos. 2003. A framework for classifying denial of service attacks. In: *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications,* ACM, pp. 99–110.

[91] M. Geva, A. Herzberg and Y. Gev. 2014. Bandwidth distributed denial of service: Attacks and defenses. In: *IEEE Security & Privacy,* vol. 12, no. 1, pp. 54–61.

[92] S. Sanfilippo. 2006. *hping.* Software. Available: http://www.hping.org/.

[93] A. Kleen. 2017. *raw - Linux IPv4 raw sockets.* Web Document. Retrieved 26.11.2018. Available: http://man7.org/linux/man-pages/man7/raw.7.html.

[94] G: Lyon. 2011. *Port Scanning Techniques.* Web Document. Retrieved 27.11.2018. Available: https://nmap.org/book/man-port-scanning-techniques.html.

[95] spider.io. 2013. *Discovered: Botnet Costing Display Advertisers over Six Million Dollars per Month.* Web Document. Retrieved 29.11.2018. Available: http://www.spider.io/blog/2013/03/chameleon-botnet/.

[96] P. Paganini. 2016. *Linux/Mirai ELF, when malware is recycled could be still dangerous.* Web Document. Retrieved 29.11.2018. Available: http://securityaffairs.co/wordpress/50929/malware/linux-mirai-elf.html.

[97] CERT Division. 1996. *1996 CERT Advisories - CA-1996-21: TCP SYN Flooding and IP Spoofing Attacks.* Web Document. Retrieved 29.11.2018. Available: https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=496170.

[98] JavaPipe. 2018. *35 Types of DDoS Attacks Explained.* Web Document. Retrieved 29.11.2018. Available: https://javapipe.com/ddos/blog/ddos-types/.

[99] S. Savage, et al. 1999. TCP congestion control with a misbehaving receiver. In: *ACM SIGCOMM Computer Communication Review,* vol. 29, no. 5, pp. 71–78.

[100] Microsoft. 2012. *SQL Injection.* Web Document. Retrieved 29.11.2018. Available: https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms161953(v=sql.105).

[101] R. Barnett. 2011. *ModSecurity Advanced Topic of the Week: Mitigating Slow HTTP DoS Attacks.* Web Document. Retrieved 29.11.2018. Available: http://blog.spiderlabs.com/2011/07/advanced-topic-of-the-week-mitigating-slow-http-dos-attacks.html.

[102] V. Paxson. 2001. An analysis of using reflectors for distributed denial-of-service attacks. In: *ACM SIGCOMM Computer Communication Review,* vol. 31, no. 3, pp. 38–47.

[103] J. Mogul. 1984. *BROADCASTING INTERNET DATAGRAMS IN THE PRESENCE OF SUBNETS.* RFC 922.

[104] M. Şimşek and A. Şentürk. 2018. Fast and lightweight detection and filtering method for low-rate TCP targeted distributed denial of service (LDDoS) attacks. In: *International Journal of Communication Systems,* e3823.

[105] Q. Yan and R. F. Yu. 2015. Distributed denial of service attacks in software-defined networking with cloud computing. In: *IEEE Communications Magazine,* vol. 53, no. 4, pp. 52–59.

[106] N. Agrawal and S. Tapawsi. 2017. Defense schemes for variants of distributed denial-of-service (ddos) attacks in cloud computing: A survey. In: *Information Security Journal: A Global Perspective,* vol. 26, no. 2, pp. 61–73.

[107] K. J. Higgins. 2008. *Permanent Denial-of-Service Attack Sabotages Hardware.* Web Document. Retrieved 30.11.2018. Available: https://web.archive.org/web/20081208002732/http://www.darkreading.com/security/management/showArticle.jhtml?articleID=211201088.

[108] C. Glenn, et al. 2006. Denial-of-service attack-detection techniques. In: *IEEE Internet computing,* vol. 10, no. 1, pp. 82–89.

[109] D. J. Bernstein. 1996. *SYN cookies.* Web Document. Retrieved 2.12.2018. Available: http://cr.yp.to/syncookies.html.

[110] FreeBSD. 2008. *FreeBSD Kernel Interfaces Manual - SYNCACHE(4).* Web Document. Retrieved 2.12.2018. Available: https://www.freebsd.org/cgi/man.cgi?syncookies.

[111] L. Schabel. 2018. *Working with SYNPROXY.* Web Document. Retrieved 2.12.2018. Available: https://github.com/firehol/firehol/wiki/Working-with-SYNPROXY.

[112] A. D. Keromytis, V. Misra and D. Rubenstein. 2004. SOS: An architecture for mitigating DDoS attacks. In: *IEEE Journal on selected areas in communications,* vol. 22, no. 1, pp. 176–188.

[113] C. Kustarz, et al. 2016. *System and method for denial of service attack mitigation using cloud services.* U.S. Patent: 9432385.

[114] D. Farinacci, et al. 2000. *Generic Routing Encapsulation (GRE).* RFC 2784.

[115] L. H. Holloway, et al. 2017. *Identifying a denial-of-service attack in a cloud-based proxy service.* U.S. Patent: 9628509.

[116] G. Pallis and A. Vakali. 2006. Insight and perspectives for content delivery networks. In: *Communications of the ACM,* vol. 49, no. 1, pp. 101–106.

[117] D. Moore, et al. 2006. Inferring internet denial-of-service activity. In: *ACM Transactions on Computer Systems (TOCS),* vol. 24, no. 2, pp. 115–139.

[118] H. Welte and P. N. Ayuso. 2014. *Documentation about the netfilter/iptables project.* Web Document. Retrieved 5.12.2018. Available: https://netfilter.org/documentation/index.html#documentation-faq.

[119] S. Watkiss. 2018. *Introduction to Linux security principles.* Web Document. Retrieved 5.12.2018. Available: http://www.penguintutor.com/linux/introduction-linux-security.

[120] Python Software Foundation. 2018. *python.* Software. Available: http://www.python.org/.

[121] R. Halley. 2018. *dnspython.* Software. Available: https://github.com/rthalley/dnspython.

[122] J. L. Santos, et al. 2013. Implementing NAT traversal with private realm gateway. In: *Communications (ICC), 2013 IEEE International Conference on,* IEEE, pp. 3581–3586.

[123] A. Koskimäki. 2018. *Custom DNS relay module for use with Realm Gateway software.* Software. Available: https://github.com/Aalto5G/CustomDNSRelayForRGW

[124] D. Thaler and C. Hopps. 2000. *Multipath Issues in Unicast and Multicast Next-Hop Selection.* RFC 2991.

[125] H. Kabir, J. L. Santos and R. Kantola. 2016. Securing the Private Realm Gateway. In: *Networking,* pp. 243–251.

[126] A. Sing. 2004. *An Introduction to Virtualization.* Web Document. Retrieved 9.12.2018. Available: http://www.kernelthread.com/publications/virtualization/.

[127] Oracle Corporation. 2018. *VirtualBox.* Software. Available: https://www.virtualbox.org/.

[128] Oracle Corporation. 2018. *VirtualBox Manual - Chapter 6. Virtual networking.* Web Document. Retrieved 10.12.2018. Available: https://www.virtualbox.org/manual/ch06.html.

[129] Cisco. 2015. *Bridging and Switching Basics.* Web Document. Retrieved 10.12.2018. Available: http://docwiki.cisco.com/wiki/Bridging_and_Switching_Basics.

[130] Arch Linux. 2018. *chroot.* Web Document. Retrieved 12.12.2018. Available: https://wiki.archlinux.org/index.php/Chroot.

[131] M. Kerrisk. 2018. *namespaces - overview of Linux namespaces.* Web Document. Retrieved 12.12.2018. Available: http://man7.org/linux/man-pages/man7/namespaces.7.html.

[132] M. Kerrisk. 2018. *cgroups - Linux control groups.* Web Document. Retrieved 12.12.2018. Available: http://man7.org/linux/man-pages/man7/cgroups.7.html.

[133] Linux Containers. 2018. *Linux Containers.* Software. Available: https://linuxcontainers.org/.

[134] W. Felter, et al. 2015. An updated performance comparison of virtual machines and linux containers. In: *Performance Analysis of Systems and Software (ISPASS), IEEE International Symposium On.* IEEE, pp. 171–172.

[135] G. Giacobbi. 2006. *The GNU Netcat.* Software. Available: http://netcat.sourceforge.net/.

[136] OpenBSD Project. 2018. *OpenSSH.* Software. Available: https://www.openssh.com/.

[137] G. Scrivano, et al. 2017. *GNU Wget.* Software. Available: https://www.gnu.org/software/wget/.

[138] NGINX Inc. 2018. *NGINX.* Software. Available: https://www.nginx.com/.

[139] Microsoft. 2014. *NET: DNS: DNS client resolution timeouts.* Web Document. Retrieved 28.12.2018. Available: https://support.microsoft.com/en-us/help/2834226/net-dns-dns-client-resolution-timeouts.

[140] M. Kerrisk. 2017. *Linux Programmer's Manual - RESOLV.CONF(5).* Web Document. Retrieved 28.12.2018. Available: http://man7.org/linux/man-pages/man5/resolv.conf.5.html.

[141] J. M. Tilli. 2017. *ldpairwall.* Software. Available: https://github.com/Aalto5G/ldpairwall.

[142] R. Housley, et al. 1999. *Internet X.509 Public Key Infrastructure Certificate and CRL Profile.* RFC 2459.

[143] D. Cooper, et al. 2008. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* RFC 5280.

[144] T. Aura and A. Paverd. 2018. *Public Key Infrastructure & Web Security.* Lecture, CS-C3130 Information security, Aalto University.

[145] E. Rescorla and N. Modadugu. 2012. *Datagram Transport Layer Security Version 1.2.* RFC 6347.

[146] T. Dierks and E. Rescorla. 2008. *The Transport Layer Security (TLS) Protocol Version 1.2.* RFC 5246.

[147] A. Ajitsaria. 2018. *What is the Python Global Interpreter Lock (GIL)?.* Web Document. Retrieved 31.12.2018. Available: https://realpython.com/python-gil/.

[148] K. Mikoluk. 2013. *Python vs C: A Beginner's Guide.* Web Document. Retrieved 31.12.2018. Available: https://blog.udemy.com/python-vs-c/.

[149] pypy.org. 2018. *PyPy.* Software. Available: https://pypy.org/index.html.

[150] Nominum, Inc. 2018. *dnsperf(1) - Linux man page.* Web Document. Retrieved 31.12.2018. Available: https://linux.die.net/man/1/dnsperf.

A Appendix - RIP and OSPF routing details

This appendix presents briefly more details about RIP and OSPF routing, where differences were summarized earlier in Table 2. Simplified RIP routing procedure is presented in Figure A1 for a small example network topology. The RIP routing table information is propagated by each node retrieving the current routing table data from neighbors with UDP request and response messages. Router's routing table is then updated based on this data which can lead to the router adding new entries to its table as it gets topology data further and further from the network until eventually the whole network topology has converged to each router, where hop count is used to measure distances. As this update process is done in a step-by-step manner, the converging times may be high depending on the network size and the interval of update messaging.[3, pp. 142–194][33]

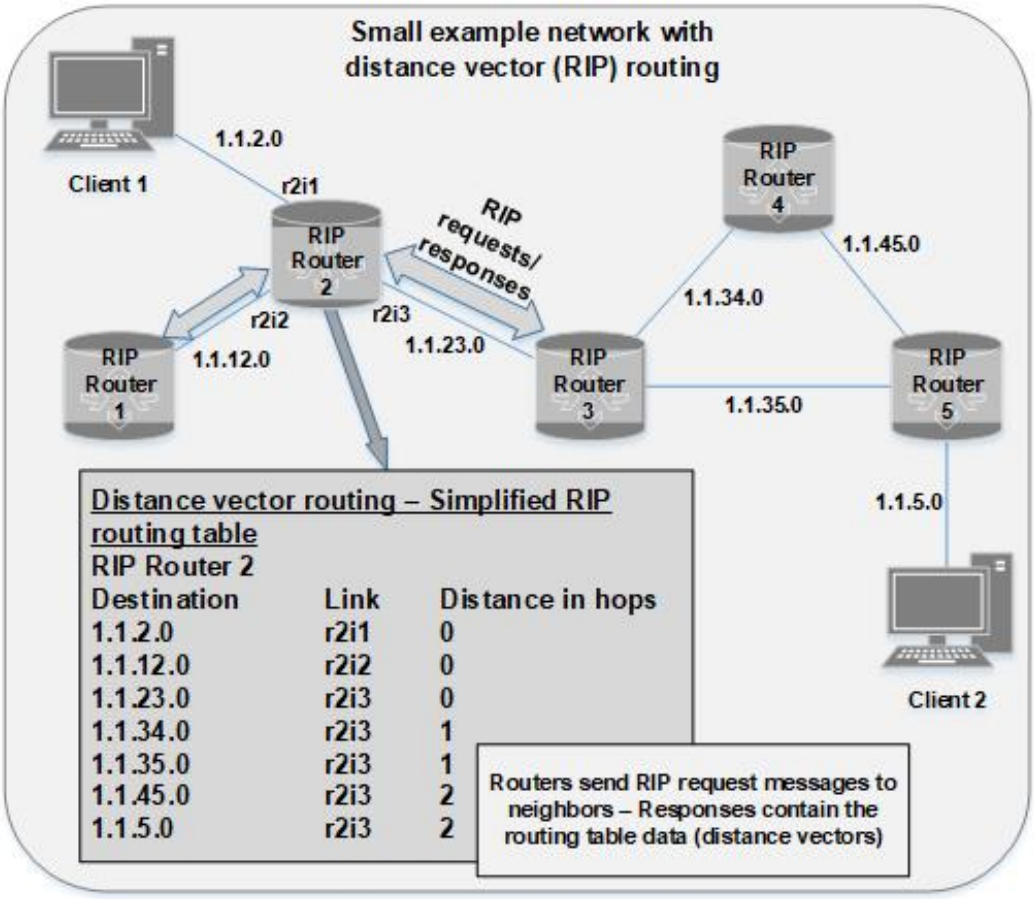

Figure A1: RIP routing principles

The major limiting for factor using distance vectors is the counting-to-infinity problem which may occur when there are sudden changes to the network topology. Simple example in Figure A2 illustrates this issue, where routers R2 and R3 are suddenly disconnected. Router R2 does notice the malfunctioning link and sets the destination

inaccessible but at the same time router R1 still maintains that the distance towards R3 is 2 hops through R2 which it then conveys to R2 which updates its routing table accordingly. Router R2 now thinks that R3 is is available through R1 and this is conveyed to R1. This back-and-forth messaging continues where each step increases the hop count in R1 and R2 by 1 towards infinity if the process is left to be. RIP does take this problem into account and initial defense against this issue is setting the infinity value to 16 hops, so the nodes don't add hop values beyond that. Additionally, RIP employs methods called split horizon and route poisoning which limit the advertisement of routes back from one note to another and set the malfunctioning routes to be advertised as infinite from the get-go respectively.[3, pp. 56–82][33]

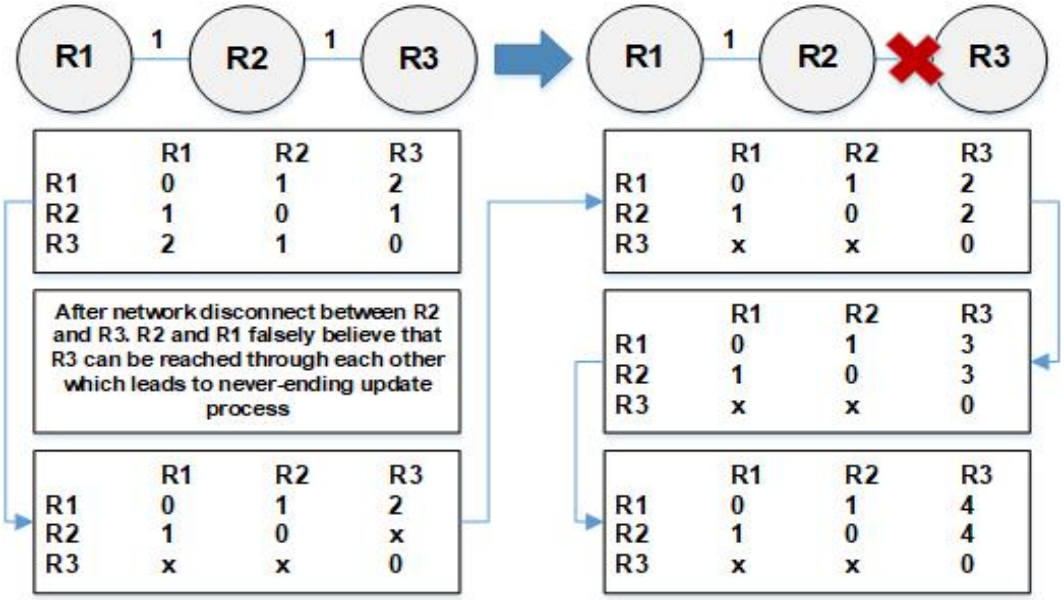

Figure A2: Distance vector routing and counting to infinity

For comparison, the OSPF routing procedure in simplified form is presented in Figure A3, again for a small example network topology. Here each node maintains a separate link state database which is common for all routers in the network and is set up when the network is initialized. The routing table is then calculated from this database by the Dijkstra's open shortest path first algorithm, where the resulting table, which is specific for each router, is used for the routing decisions. For updating data about the network topology, only changes to the link state database are communicated by broadcasting update messages to all routers in the network, when, for example, a router notices that a link has gone down. OSPF doesn't actually use TCP or UPD for signaling but employs IP in more direct manner similarly to ICMP. For IPv4, the protocol messaging is packaged straight to IP packets with the IP header protocol field denoting that the message is for OSPF.[3, pp. 142–194][34]

As was presented in Table 2, there are clear reasons for using OSPF over RIP with

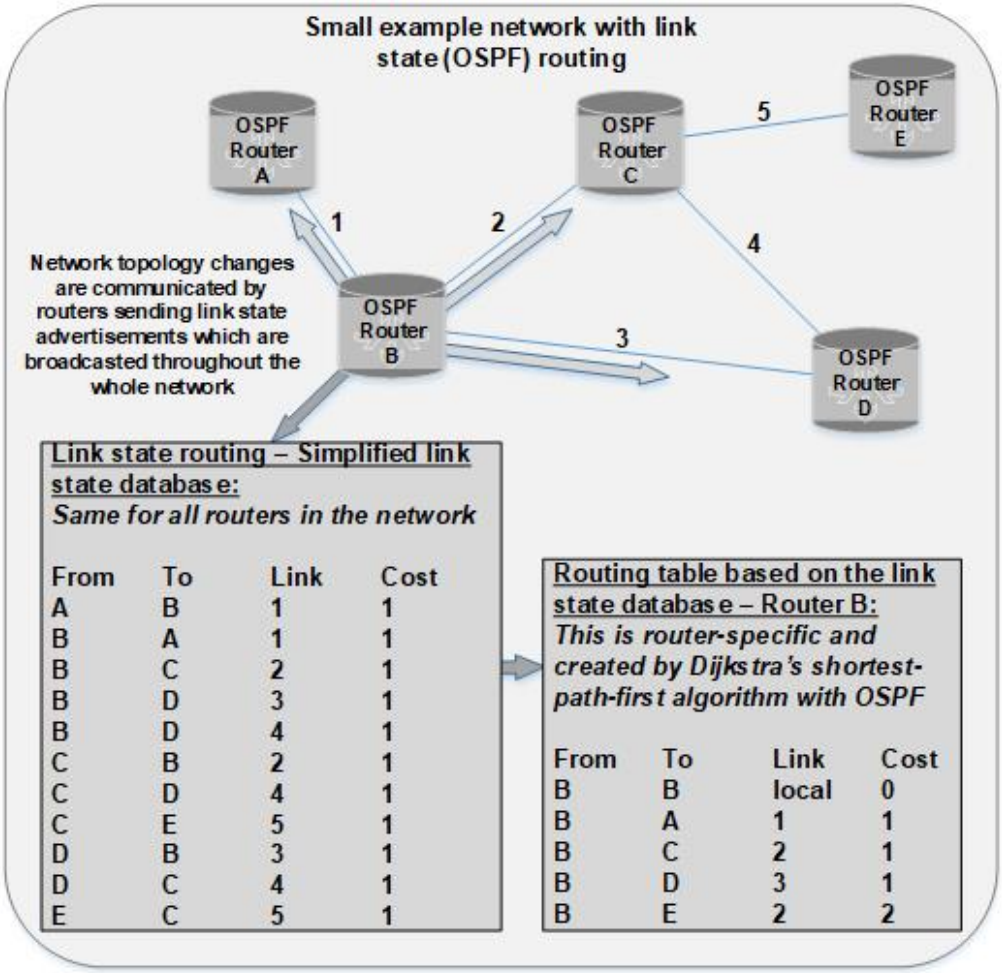

Figure A3: OSPF routing principles

the cost of having a more complex protocol in use. The first major benefit is the avoidance of loops as each OSPF router has access to the whole network topology all the time and can then calculate routes avoiding broken links. Additionally, the second benefit is the capability to use link costs for denoting route quality for making more informed routing decisions with QoS in mind. Finally, there is a benefit of faster routing table updating time as network changes, due to the link state advertisement broadcasts instead of sending updates hop-by-hop.[3, pp. 142–194]

With both RIP and OSPF, the main issue is scalability, though. RIP is limited by the hop count, whereas OSPF is limited by the resource demands for handling the link state update broadcasts and for calculating the shortest path with Dijsktra's algorithm for large networks. OSPF has a workaround for enabling some increases to network size with utilizing hierarchical structure from which there is an example in Figure A4, where the network is divided to sub-networks. The idea is to aggregate route information utilizing the CIDR network masking with the area border routers to keep the link state databases small and to limit the amount of update message traffic.

In the example, networks in Area 1 and Area 2 see only 1 route entry to outside which are the respective border router gateways, which then forward outbound traffic towards the correct destination outside their internal networks. This works for the other direction too as the area border routers can aggregate their internal network information to fewer routing table entries which are then communicated to their routing neighbors outside. The area concept is essentially expanded to Autonomous Systems (AS), as can be seen in the sections describing BGP in Chapter 2.[34]

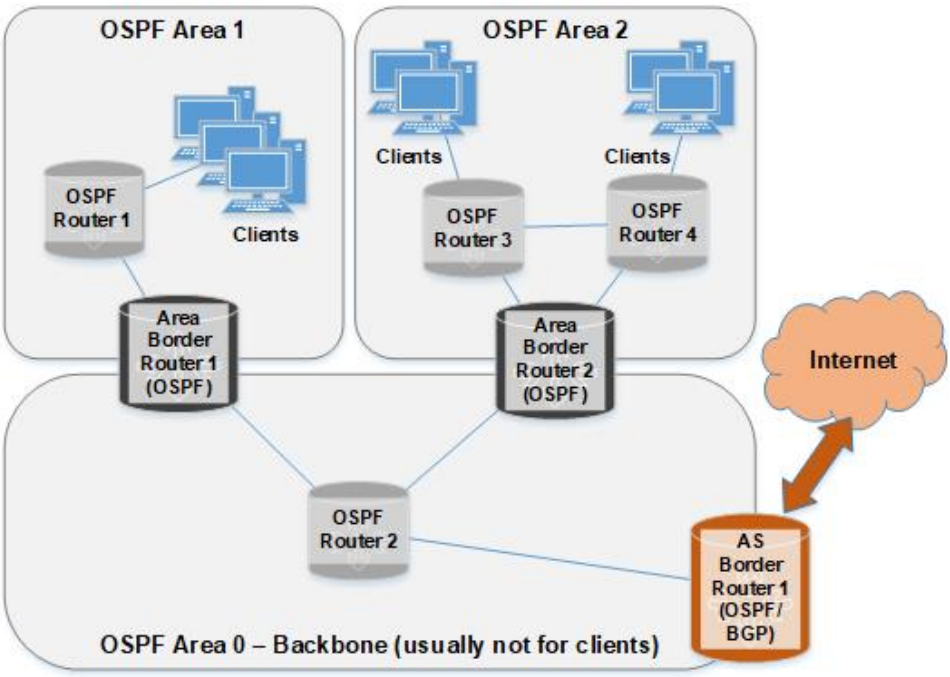

Figure A4: OSPF areas example

B Appendix - PKI and TLS principles

This appendix presents the principles of PKI and TLS, where the idea behind public and private keys is explored further. Additionally, there is an overview on what the PKI key distribution process involves. For a start, an example for public-key cryptography algorithm is the *Rivest–Shamir–Adleman (RSA)*, where the asymmetry or the challenge to solve the private key if public key is known is based on the difficulty of factorizing the product of two large prime numbers. In a more formal manner, the RSA basis is

$$(m^e)^d \equiv m \pmod{n}, \qquad 0 \leq m < n. \tag{B1}$$

In the previous equation, the e, d and m can be thought as large integers, where it is difficult to solve d, even if e, m and n are known. For encryption and decryption usability, it is also good that reversing the place of d and e is possible, which implies that

$$(m^d)^e \equiv m \pmod{n}, \qquad 0 \leq m < n. \tag{B2}$$

For the key generation for RSA, the d, e and n are calculated with the help of randomly generated two random prime numbers p and q. Variable n would then be pq, which is the modulus value and can be thought as the encryption key length. After this, the e is chosen in a manner, that

$$\gcd(e, \operatorname{lcm}(p-1, q-1)) = 1, \qquad \operatorname{lcm}(p-1, q-1) = \lambda(n), \tag{B3}$$

Where lcm is the least common multiple or the smallest common integer which is divisible by the two given integer values, and where the gcd is greatest common divisor or the largest integer which divides both of the given values. The lambda-function here would denote Carmichael's totient function with the n as input. This would then lead to

$$d \equiv e^{-1} \pmod{(\lambda(n))}, \tag{B4}$$

where solving d can be a long and problematic iterative process, if the key generation parameters are chosen properly. The encryption scheme with public key e would then be

$$c \equiv m^e m \pmod{n}, \qquad 0 \leq m < n, \tag{B5}$$

so that encrypted text c is attained with plaintext message turned to integer with using padding, where this adjusted message is denoted by m here. The process of padding means that agreed upon, reversible padding scheme fills the message to a certain length, so that "empty space" is not encrypted as this would help in cracking the encryption scheme. The message data can then be decrypted by solving

$$c^d \equiv (m^e)^d \equiv m \pmod{n}. \tag{B6}$$

Additionally, the signing of messages is based on

$$(h^e)^d = h^{ed} = h^{de} = (h^d)^e \equiv h \pmod{n}, \qquad h = \operatorname{hash}(m), \tag{B7}$$

where the hash denotes a suitable one-way hashing function that is used both by the sender and receiver of the signed message, and where h is the calculated hash-value from the plaintext m and the commutative properties of multiplication are utilized. The verifying signature of some message would be formed by encrypting the hash with d, in a similar manner to the case where somebody encrypts the message meant for the private key holder with e. This signature value can then be decrypted with e, where the resulting hash can be compared to the hash value that the recipient can calculate from the received message text. If this hash and the decrypted signature are the same, it means that the message was signed by the private key holder and was not changed during the transmission.[75, pp. 283–376]

One major problem with public encryption keys is the proper distribution of said keys, or rather the case of ensuring that the presented public key to some web service is connected to the actual web service and to the private encryption key it uses. If no additional measures are taken, it would be difficult for a client to contact a legitimate web service, if multiple services are on similar domains and each offers up some public key, claiming to be the real deal. To solve this, the PKI utilizes a Certificate (CERT) concept based on the X.509 standard, where a web server is issued a signed CERT by a Certificate Authority (CA), which connects the respective service identity (usually a FQDN) to a specific public key. The basic structure of a system utilizing CERTs is presented in Figure B1.[74][142][143][144]

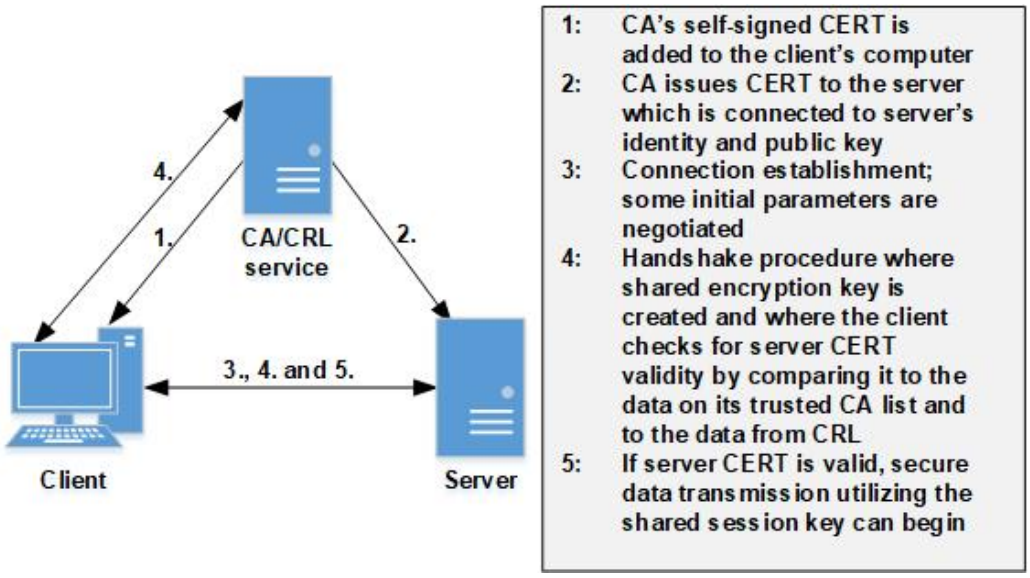

Figure B1: Basic PKI architecture

For the PKI base, the CAs form an authority chain, where the highest-level CA is a trust anchor. The trust anchor will self-sign the CERT for itself, and then, generally as a service, will sign the certificates to CAs on the level below it, where each CERT will also have a note pointing to the signer on the level above. This chain will then continue towards the actual web service, denoted by the server in Figure B1, who has

the CERT signed by the immediate CA. The trust anchors are usually well-trusted and established Internet entities that often have monetary incentives to issue CERTs in a proper and safe manner. To enable the CA validation checks to work smoothly, trust anchor CAs have usually distributed their self-signed CERTs to a large number of Internet clients either via web browser installations or with OSes.[142][143][144]

In Figure B1, the first point is that the client would have received the CA's self-signed certificate, likely upon the overall initial setup of his computer. This CERT is then found on the trusted CA list on the client, which is usually valid for a long time. The second point is the server retrieving a CERT linked to it from the CA, usually by just purchasing it for a certain, relatively long period such as for a year. Upon contacting the web service, the client will receive the server's CERT, and the client can then check the CERT signature for validity based on the note about the assigning CA in the CERT, when the respective public key is found from the client's trusted CA list. During this process, the CERT's expiry and revocation status should be checked, which is usually done from a local cache which is updated regularly by the OS from a Certificate Revocation List (CRL) database maintained usually in some manner by the CA. If these checks are passed and the CERT's subject matches the contacted service, the client can then assume that the public key in the CERT would link to the server's private key, as intended.[142][143][144]

As the client can presumably trust the service after the CERT validation, the follow-up is usually to form an encrypted connection, as sensitive information is to be transmitted. Overall, this connection establishment is a two-stage process of a handshake or session key exchange and the actual, encrypted data transfer which is referred as a session. In PKI, it is often the case that the client would first validate the server identity and if the client would need to authenticate to the server, the credentials would be given after a successful handshake step to utilize the secure connection. During the handshake, the client and the server will form a shared secret key based on their public and private encryption keys. This secret is then used as a shared encryption and decryption key for a symmetrical encryption scheme which is used to encrypt the transmitted data for the remaining duration of the session, as it is less complex to process, compared to asymmetric encryption scheme.[74][144]

For the session key exchange and session data transmission in Figure B1, protocols like *Transmission Layer Security (TLS)* come in. The newest version of TLS, which is built on top of the older SSL, is 1.3, and it is used generally in conjunction with HTTPS and over TCP, whereas UDP connections can use Datagram Transport Layer Security (DTLS) protocol [24][145]. With TLS, the aforementioned PKI handshake procedure is defined, which establishes the shared secret that is to be used with a supported cipher suite for communication encryption. The handshake, which is presented in Table B1, is based on the Diffie-Hellman key exchange protocol where the security foundations are quite similar to those of RSA, as they are both based on the difficulty on solving the discrete logarithm problem. In this table, Values x and y are the client and server public keys and the nonces refer to pseudo-random values

generated by the client and the server, where one part of validating the connection is the decryption of the server-signed nonce on step 2 with the server's public key. Additionally, message authentication code concept is used here to denote values that can be used to verify message integrity. When calculating these codes to finalize the handshake during steps 4 and 5, the shared secret S_{shared} would then be

$$S_{shared} = g^{yx} \pmod{n}, \tag{B8}$$

and with this value, the master secret S_{master} to be used with the chosen symmetric encryption scheme is

$$S_{master} = \text{hash}(S_{shared}, label, N_c, N_s), \tag{B9}$$

where pre-determined hash function is used on the client and server nonces N_c and N_s, shared secret and pre-determined text label.[22][75, pp. 515–522][144]

As was implied in Table B1, TLS offers a selection of cipher suites to be used in both the initial handshake, as the encryption cipher and to validate message integrity during the session. In addition, the encryption key strengths are also negotiated during the handshake process. TLS version 1.2, which is still widely in use, offers for example multiple versions of AES for the symmetric encryption and also multiple versions of Diffie-Hellman and RSA, some of which utilize elliptic curve cryptography for even more security. The RSA and Diffie-Hellman differ to some extent, as RSA is purely an asymmetric scheme, where Diffie-Hellman is a symmetric one with the private and public key components. In any case, TLS is thought to be a hybrid cryptosystem due to the handshake procedure and the following symmetric session encryption that generally utilize different security mechanisms. It is good to note that the previous TLS 1.2 could be made to use some insecure protocols during the parameter negotiation and one major difference with the newer TLS 1.3 is the removal of support for various, older cipher suites, where there isn't even an option to select poorly, so to speak.[22][146]

Table B1: TLS handshake process

Step	**Transmitted data**
1 - Client → Server	Client sends version data, session id, client nonce value and selection of supported cipher suites to the server
2 - Server → Client	Server responds with the version data, same session id, server nonce, selected cipher suite and its CERT with the chain going to the trust anchor; additionally, the server transmits at this point specific parameters which are used for calculating the shared secret and for validating the connection legitimacy: values g, n, q^y mod n and signatures based on both of the nonces, g, n and q^y, where g and n are primes
3 - Client → Server	Client will now send value q^x mod n to the server, accompanied by a note in changing the cipher mode to be encrypted and a final message authentication code value based on all the previous handshake protocol messaging and on the master secret, derived from the shared secret and from the nonces
4 - Server → Client	Finally, server will respond with a similar message authentication code value based in the previous session protocol messaging and the master secret and these two code values should then match
5 - Server ←→ Client	The encrypted data transfer can commence and the shared and validated master secret can then be used to encrypt end decrypt the data for this session

C Appendix - Python and C code performance comparison

Choosing an optimal programming language for high performance web service can be problematic and one has to take development times into account where it matters when working code and actual programs hit the table. There is no way around the simple fact, though, that Python is a high-level programming language whereas C (and C++) is a mid- to low-level programming language which will usually result C programs being noticeably quicker when handling massive amounts of operations. There are few technical key reasons for this performance difference as is elaborated below:

- While C code is compiled first and the run, Python code is run with the Python interpreter program which will take up additional resources.
- With C, the memory management is totally in the hands of the coder and works on a very low-level, for better or worse, whereas with Python, similarly to other high-level programming languages, the interpreter will handle memory use automatically which may lead to programs using memory in non-optimized manner.
- Python is actually built upon C and various Python primitives and programming constructs are actually C entities with added complexity. For example, just the standard integer variable in Python is a C class object with varying attributes. This can result Python code taking up more resources even with simple operations compared to C code, where at least the programming primitives are very lightweight.
- Python interpreter may do unnecessary logical checks and other code monitoring during the program run which will use additional computing resources.
- Important limitation with the Python is also Python's global interpreter lock which will prevent python threads from utilizing multiple processor cores simultaneously. Python programs need separate processes to do this which may include signaling overheads if data is to be transmitted between processes via process managers or queues.[147].[148]

To illustrate the performance differences with simple network services, a test was done using UDP services done either with C or with Python. The test setup was the same cloud system from Figure 30, that was in use with delay and DDoS-tests earlier in the thesis. In this comparison case, one cloud server would act as a client and another cloud VM would be the server. With this test, the first setup involved an UPD echo server which would echo any UDP messages sent to it back to the client. The second case utilized similar UDP server, but now small amount of textual data was retrieved from the server memory and this was then used as an UDP reply to the client. The final case involved setting up BIND DNS server to act as a C program which did hold just one address mapping in its service database that could then

be queried. For comparison, a simple, custom DNS server using UDP was set up with python, where *dnspython* library was used to parse incoming DNS queries [121], retrieve address data from memory containing just one address, and then create a DNS reply to be sent back to the client. The results of this test are shown in Figure C1.

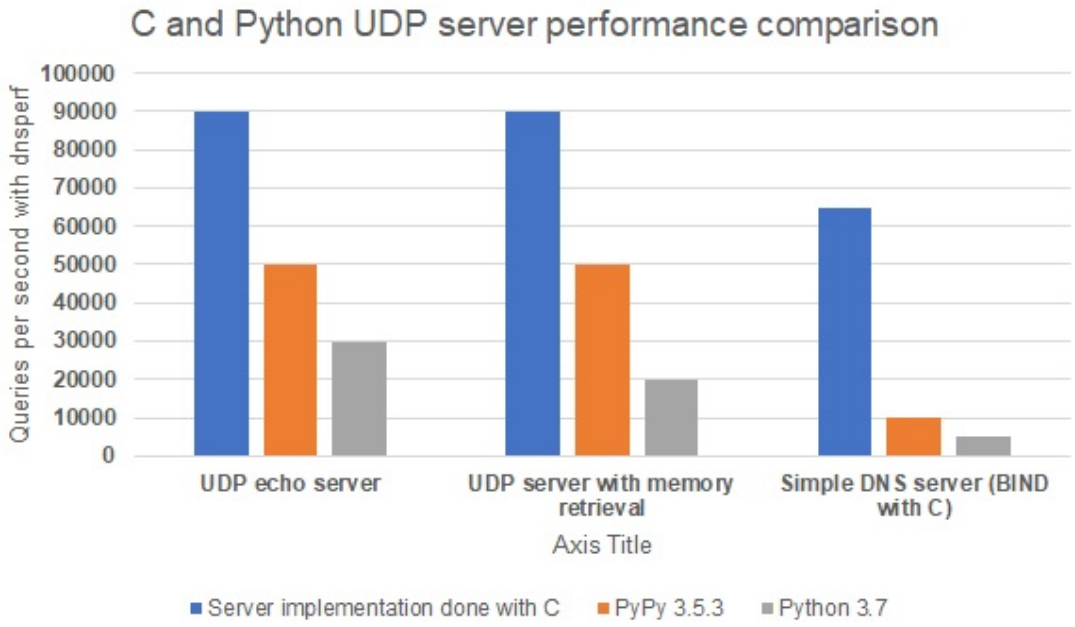

Figure C1: C and Python UDP server performance comparison

With Python, the server program was run with both Python 3.7 which is the newest stable version when writing this thesis, and with *PyPy 3.5.3*, which is an optional Python interpreter with performance optimizations [149]. The client software in this test was the Linux *dnsperf* tool that could be used to send UDP messages or UDP DNS queries at very high rates [150]. With very little memory management or object creation operations, the capacity difference is not that big, especially between PyPy interpreter and C, but problems arise when the memory manipulation and object creation operations accumulate which is an eventuality with more complex programs such as DNS server.

There are, however, clear benefits for programming with Python. For one, Python has vast amount of additional libraries which will make coding complex software far easier than doing the same with C. Additionally the syntax and lack of tedious memory management make programming with Python generally much faster. There is also the option to utilize effective C code within Python programs as has been done with various additional Python libraries such as NumPy, but this may make the program code to be very complex and cluttered. In conclusion, Program prototypes can be made much faster with Python, but for actual deployment for high-stress server programs, serious thought must be put into porting the program to more efficient lower-level programming languages such as C or C++.