NTNU – Trondheim
Norwegian University of
Science and Technology

Deep Packet Inspection Bypass

Thomas Kjøglum

Master of Science in Communication Technology
Submission date: February 2015
Supervisor: Stig Frode Mjølsnes, ITEM

Norwegian University of Science and Technology
Department of Telematics

1

Title: Deep Packet Inspection Bypass

Student: Thomas Kjøglum

Problem description:

Authoritarian governments consistently request the network operators to censor internet communications. Deep packet inspection systems and tools are regularly in use for this purpose. Several techniques have been proposed to bypass these communication restrictions. One example is the Kickstarter project titled "Operator, a News Reader that Circumvents Internet Censorship" by Brandon Wiley. Wiley has designed a protocol "Dust" [1] that aims to defeat a number of filtering methods currently in active use to censor Internet communication.

Some basic questions are: How is it possible to bypass deep packet inspection filters with high likelihood? On the other hand, could the approach of Dust and other filtering bypass techniques be useful for masquerading malicious code?

The candidate will start out by investigating possible techniques for bypassing a open source packet inspection tool, such as SNORT. The candidate will support his experimentation by method of setting up and running hacker competitions (or trials) where the participants' challenge will be to set up, configure and run efficient deep packet inspection systems directed against various types of "subversive communications" generated by the organizer of the competition.

[1] WILEY, Brandon. Dust: A blocking-resistant internet transport protocol. Technical report. http://blanu.net/Dust.pdf, 2011.

Responsible professor: Stig Frode Mjølsnes, ITEM

Supervisor: Stig Frode Mjølsnes, ITEM

Abstract

Internet censorship is a problem, where governments and authorities restricts access to what the public can read on the Internet. They use deep packet inspection tools to conduct the censorship. An example of such a tool is Snort. Snort is a signature-based network inspection tool that sits on the edge of a network which is monitored and inspects packets that passes through.

To avoid Internet censorship it is possible to bypass the deep inspection tools, and it is multiple ways of doing this. Methods that may be used depends on in which network layer the Internet censorship is done. This is mainly done by changing the traffic appearance too not match the rules written in the signature-based inspection tool. There are multiple software solutions already proposed that are capable of doing this, two of them are Snort and Pluggable Transport. These two solutions are capable to masquerade network traffic to look like benign traffic, and it is not possible to single out the traffic flow from other traffic flow.

It is possible to use these two software solutions to transfer malicious code to a company network. This requires that a person inside the company network downloads the malicious code using the software. Malicious code which is downloaded using either Dust or Pluggable Transport software would not be detected, they are designed to look like normal Internet traffic.

Sammendrag

Internettsensur er et problem, det eksisterer regjeringer som setter begrensninger for hva allmennheten får lov til å se på Internett. Disse regjeringene bruker pakke-inspeksjons-verktøy som kan se inn i nettverkspakker, eksempel på et slikt verktøy er Snort. Snort er et signaturbasert verktøy som befinner seg på kanten av et overvåket nettverk og analyserer all trafikk som passerer.

Det er mulig å unngå Internettsensur ved å unnvike disse inspeksjons-verktøyene som kan se inn i nettverkspakker. De eksisterer mange mulig måter å gjøre dette på, men det avhenger av hvor i OSI modellen sensuren er gjennomført. I hovedsak handler det om a endre trafikken slik at den ikke passer med reglene som er skrevet i den signaturbaserte inspeksjons-verktøyet. Det eksisterer flere program som løser akkurat dette, to eksempler er Snort og Pluggable Transport. Disse programmene kan maskere trafikk som ikke er lovlig til å ligne på lovlig trafikk, det er da umulig å se forskjell på den maskerte og ikke maskerte trafikken.

Snort og Pluggable Trasport kan også brukes til å frakte ondsinnet kode inn i et bedriftsnettverk, men dette krever at en person eller programvare på innsiden av bedriftens nettverk laster den ned. Hvis slik kode blir lastet ned ved hjelp av enten Dust eller Pluggable Transport ville det ikke vært mulig å se, dette på grunn av at disse programvarene er laget for å se ut som vanlig Internett trafikk.

Preface

This topic evolved from a number of talks with my Professor Stig Frode Mjølsnes, at the start we were focusing on finding a topic which could build upon my previous project. This project gave a recommendation on how to conduct a hacker competition here at Gløshaugen, NTNU. This is the reason why it says "the candidate will support his experimentation by method of setting up and running hacker competitions (or trials)" in this project description.

Stig Frode had recently read about a new protocol named Dust and handed the paper to me. After reading about on it, and figured out that its focus on Internet censorship, I wanted to learn more. This was the beginning of the problem description.

I would like to express my gratitude to my Professor and supervisor Stig Frode Mjølsnes for the discussions and help with the thesis. Also for the help in finding relevant studies and solutions regarding Internet censorship.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1
Introduction

In many places around the world there exists people who are victims of Internet censorship, where governments and authorities censor information that they believe the public should not have access to. The information which is censored may be social media, political content, laws and regulations, blogs, and information on how to circumvent Internet censorship. The examples of information which may be censored is from the fourth report in a series of comprehensive studies of Internet freedom around the world. The name of this study is Freedom on the Net 2013 [FH13], it covers development regarding Internet freedom in 60 countries. The report shows Internet censorship worldwide is declining, with 34 out of 60 countries experienced a negative trajectory during the overall coverage period from May 2012 to April 2013. USA Today wrote an article that lists the top ten Internet censored countries [UT14], where it is the governments and authorities which censors. The country that tops the list is North Korea, where the government controls all websites and only 4 percent of the population has access to the Internet.

To avoid Internet censorship it is necessary to bypass the filtering mechanisms which is used, this work will investigate how this is possible using the Snort software as a case study. It will also investigate if it is possible to use known solutions to masquerade malicious code, and then bypass DPI filters.

The threat model for bypassing a filtering mechanism is a government who is trying to block an individual from retrieving content from the external network, this threat model is shown in Figure 1.1. This scenario is only appropriate when trying to avoid internet censorship and bypassing DPI filters. This threat model is not applicable when trying to masquerade malicious code. A government will probably not have any risk when letting malicious code out of their network, thus the threat model must change. When looking at the scenario of masquerading malicious code, the threat model will therefor be the Internet trying to send malicious code to a company network. In this threat model the company have a Network Intrusion Detection System (NIDS) at the edge of their internal network. This threat model is

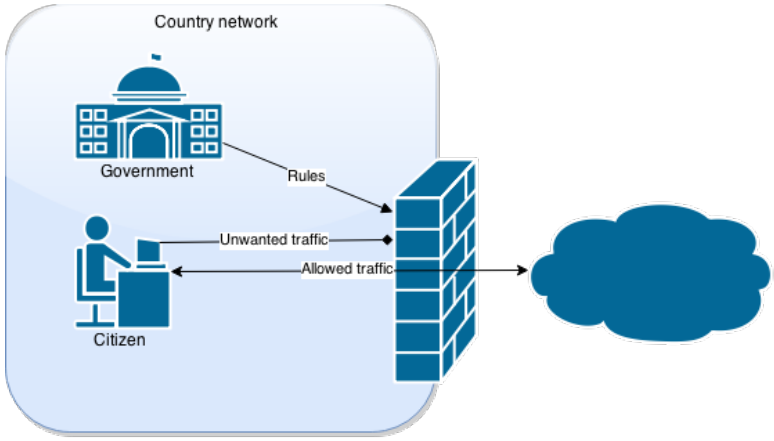

Figure 1.1: Threat model for bypassing Internet censorship filters

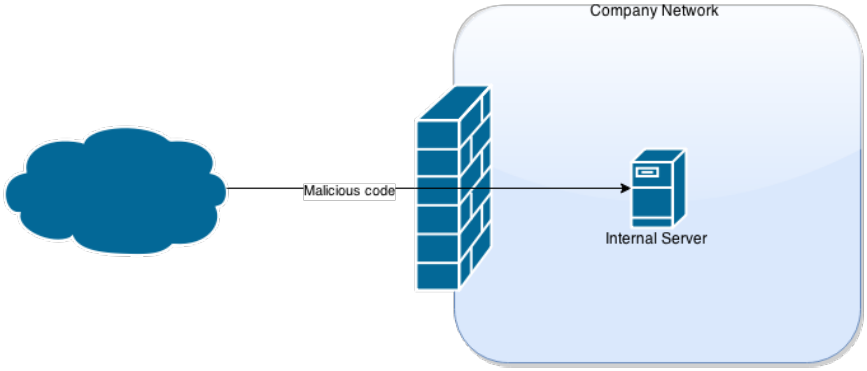

Figure 1.2: Threat model for masquerading malicious code through a DPI filter

shown in Figure 1.2.

This work will also describe what a NIDS is and how it detects unwanted traffic in the network, and present a specific NIDS tool named Snort. In Chapter 5 Snort will be used to see if it is possible to bypass the rules written in it. In Chapter 4 some theoretical ways to filter network traffic and methods to bypass them will be presented. The different filtering mechanisms and bypassing methods are presented in the relevant OSI model layer. Some known Internet censorship avoidance solutions will be presented, and they will be the bases of the discussion on possibility to masquerade malicious code to avoid NIDS.

In the project description of the thesis it says that the experiment will be supported with a hacker competition or trials, where the participants' challenge will be to set up,

configure and run efficient deep packet inspection systems directed against various types of "subversive communications" generated by the organizer. This was not carried out because of the complexity of setting up and configuring such deep packet inspection systems, it would have taken a long time for the participants to familiarized them self with the configuration and rule writing. In Chapter 7 different competition architectures are proposed.

1.1 Related Work

Brandon Wiley has started a project which he calls Operator [Wil15], this is a Rich Site Summary (RSS) news reader which will circumvent Internet censorship. The Operator project uses a protocol named Dust, also developed by Wiley, this protocol will be discussed further in Chapter 6. The Tor project has experienced filtering of their Tor traffic and has therefor been looking into censoring bypassing, which they call Pluggable Transport. A specific protocol of the Pluggable Transport will be looked at in Chapter 6.

Chapter 2
Network Intrusion Detection System (NIDS)

This chapter will a give short introduction to what a NIDS is and how it discovers unwanted traffic in the network it is supervising.

2.1 What is a NIDS

A NIDS is an Intrusion Detection System (IDS) that scans the data flowing in a network, an alternative to NIDS is a Host Intrusion Detection System (HIDS). A HIDS is an IDS agent which is installed on a host and detects attacks to this specific host. The agent often use a combination of signatures, rules, and heuristics to identify attacks and unwanted traffic. It operates as a passive agent only gathering, identifying, logging, and alerting. A NIDS is usually installed in a important network node where most of the traffic passes through. It uses a network tap, span port, or hub to collect traffic that traverse the network. The collected traffic is then analyzed, and as the HIDS, the NIDS role also is to only gather, identify, log, and alert suspicious traffic. An IDS do not actively block network traffic, it observes. [Reh03] [Ber]

2.2 How do NIDS locate unwanted traffic

Intrusion detection is to detect suspicious activity in the supervised area. There are two ways a IDS detect suspicious activity, signature-based detection and anomaly detection. Examples of signature-based IDSs is Snort and Bro, examples of anomaly-based IDSs is SPADE, ADAM and NIDES. Each of the two detection methods are further described below. [Mah03]

2.2.1 Signature-based Detection

A signature-based IDS uses hand written rules to detect unwanted traffic and behavior. It collects data packets and analyze them. If the packets have contents or signatures which matches the written rules the packet may be flagged or an alert may be raised.

To be able to do this the detection system needs a set of signatures and rules. When installing Snort a set of default rules and signatures are included.

A disadvantage of the signature-based IDS is that it needs the signature database to be up to date. All new attacks and malicious code which is unknown to the IDS will pass through it without raising an alert. This means that all zero-day vulnerabilities will not be detected, and they may be undetected for a period of time after they have been known because of the update time of the signatures and rules in the IDS

2.2.2 Anomaly Detection

A datagram packet sent on the network is of some standard form. If a packet do not have the standard form or if it contains some errors in the packet header, an anomaly-based IDS will notice this and raise a concern towards this packet. [Reh03]

An anomaly-based detection system models traffic, usually the distribution of IP addresses and ports, to learn how "normal" traffic looks like. Hostile traffic often falls outside this distribution. One advantage of the anomaly-based detection is that no rule writing is needed, this removes long zero day windows where harmful traffic may go undetected. A problem with this detection method is that it may generate false alarms. Benign traffic which deviate from normal traffic may occur, this may cause system administrators to block legit users or traffic from the network. [Mah03]

Chapter 3
Snort

This chapter will describe Snort, how it detects unwanted traffic, and how Snort rules are written. This chapter is mainly based on the content of the Snort manual. [Cis14]

3.1 Overview

Snort is a network packet sniffer and an open source lightweight NIDS. It logs packets based on rules, these rules are crafted in such a maner that it is possible to perform content pattern matching and detect a variety of attacks and other unwanted traffic. Snort offer real time alerting, with multiple alternatives for logging. Because of the rule-based content-pattern matching Snort is a IDS in the category of Signature-based detection systems.

Snort is able to decode the Application Layer content of a network packet and check whether the content matches with a rule it has loaded at start up. With this feature Snort is able to detect many types of hostile activities, including buffer overflows, Common Gateway Interface (CGI) scans, or any other data in the packet payload that can be characterized in a unique detection fingerprint. [R+99]

3.2 Modes

As mentioned in the first section, Snort is both a network packet sniffer and an open source lightweight NIDS. Additional to these modes there is one more, the packet logger mode. Both the NIDS mode and the packet logger mode are depended on the network packet sniffer mode to work. The logging mode needs packets from the network packet sniffer mode to have something to log, and the NIDS mode needs packets to analyze against its rules.

The packet sniffer mode has three switches which may be used. these are; printing Transmission Control Protocol (TCP)/IP packet headers to screen, display the IP

and TCP/UDP/ICMP headers only, and display the packet data as well as the headers.

The logger mode is how Snort should log the packets which are captured by the packet sniffer mode. The most basic is just logging it to a directory. By setting the home network switch, Snort only logs traffic which is of relevance to the specified network. If Snort is running in a high speed network the binary mode option should be used, this logs packets to a single binary file. The binary mode do not take the home network mode into account when it logs packets, it logs everything regardless of what is specified.

When using the NIDS mode, Snort requires a configuration file. This configuration file is delivered with default configuration when installing Snort. In addition there is one switch which may be specified, this is how Snort NIDS mode should display its output. It has six different output definitions; fast, full, unsock, none, console, and cmg. The fast option uses a simple format with a timestamp, alert message, source and destination IPs and ports. The full option is default if nothing is chosen. The unsock option sends alert to a UNIX socket that another program could listen on. The none option turns off alerting. The console has the same simple format as the fast option, but it prints the alert to console instead of logging them to a file. The cmg option generates cmg style alerts.

3.3 Configuration

When configuring Snort it is possible to include files and set variables. The variables may be a list, an integer or a string. How to configure Snort will not be discussed any further, but some important functionality will be mentioned. [Cis14, Chapter 2]

In Snort version 1.5 a new functionality were introduced, preprocessors. The preprocessors allow integration of modular plugins created by users and developers. The preprocessor code is executed before the detection engine is called but after the packet has been decoded. Packets may be modified or analyzed in an out-of-band manner using this integration mechanism. At this moment there are twenty four different preprocessor modules, including; Hypertext Transfer Protocol (HTTP) Inspect, SSL/TLS.

The HTTP inspect module is a generic HTTP decoder for user applications. It can decode a data buffer, find HTTP fields, and normalize these fields.

The SSL/TLS module analyzes SSL and TLS traffic and optionally determines if and when Snort should stop inspecting the encrypted traffic. Snort should ignore encrypted traffic because of performance reasons and to reduce false positives. The

module could be set to only inspect the SSL handshake of each connection and then ignore the rest of the encrypted traffic.

3.4 Writing Rules

Snort rules are divided into two logical sections, the rule header and the rule options. The rule header contains the rule's action, protocol, source and destination. The source and destination is a set of both the IP addresses and port number, where the IP address could be a single IP address or a network range using the Classless Inter-Domain Routing (CIDR) notation. The rule option section contains alert messages and information on which parts of the packet that should be inspected to determine if an alert should be raised. [Cis14, Chapter 3]

3.4.1 Rule Header

The rule header contains seven different place holders, these placeholders and their positions are show in Listing 3.1. There are eight different actions, but it is possible to define new rule types and use these as action types. In Listing 3.2 an example of a header is given. In this example the action is set to alert.

```
[Actions] [Protocols] [IP Addresses] [Port Numbers]
        [Direction] [IP Addresses] [Port Numbers]
```

Listing 3.1: Snort Rule Header placeholders

```
alert tcp 10.0.0.0/24 any -> any 1024:
```

Listing 3.2: Snort Rule header example

The eight action types already defined are:

1. alert - generate an alert using the selected alert method, and then log the packet
2. log - log the packet
3. pass - ignore the packet
4. activate - alert and then turn on another dynamic rule
5. dynamic - remain idle until activated by an activate rule , then act as a log rule
6. drop - block and log the packet
7. reject - block the packet, log it, and then send a TCP reset if the protocol is TCP or an ICMP port unreachable message if the protocol is UDP.

8. sdrop - block the packet but do not log it.

The protocols field is a placeholder for different protocols. The placeholder could be any protocol which is used to transfer data as mentioned in Chapter 4, however Snort does only support TCP, UDP, ICMP and IP at this time. In the example in Listing 3.2 the protocol field is set to TCP.

The first IP address field is for incoming traffic, also known as sender/source. Only single IP addresses, or the wildcard value *any*, or IP address ranges using the CIDR notation is supported. After the incoming IP address field comes incoming port number. This port number may be a specific port, a range of ports or any port. A range of ports have three different ways to be set, *1:10*, *:1000* and *1000:*. The *1:10* indicates ports in the range from one to ten, including one and ten. The *:1000* indicates all ports smaller than and equals to 1000, and the *1000:* indicates ports greater than and equal to 1000. In Listing 3.2 the example is alerting TCP packets which comes from the network addresses range *10.0.0.0/24* with *any* port number.

The direction operator indicates which direction that the rule applies. There are two possible direction operators -> and <>. The <> indicates that Snort should consider the address/port pairs in either the source or destination. The -> operator has the source on the left side, and the destination on the right side. In the example in Listing 3.1 the -> tells Snort that traffic from *10.0.0.0/24* to *any* IP address should be considered.

IP Addesses and *Port Numbers* on the right side of the direction operator is the destination IP address and port number. They have the same representation options as the source IP address and port number.

3.4.2 Rule Options

There are four major categories of rule options:

- general: These options provide information about the rule but do not have any affect during detection
- payload: These options all look for data inside the packet payload and can be inter-related
- non-payload: These options look for non-payload data
- post-detection: These options are rule specific triggers that happen after a rule has “fired.”

The rule options are separated using the semicolon (*;*) character, and the rule option keywords are separated from their arguments using the colon (*:*) character.

There are several options underneath each rule options category, here some of them will be mentioned. More information about rule options could be found in the Snort Manual. [Cis14, Chapter 3.3-3.7]

General Rule Options

Some general rule options are listed below:

- *msg* specifies what will be logged when an alert is raised. It is used in this format: *msg: "message text";*.
- *sid* identifies Snort rules.
- *priority* assigns a severity to rules.

Payload Detection Rule Options

Some payload detection rule options are listed below:

- *content* allows the user to set rules that search for specific content in the packet payload and trigger response based on that.
- *nocase* allows the rule writer to specify that the Snort should look for the specific pattern, ignoring case.
- *rawbytes* allows rules to look at the raw packet data.
- *depth* allows the rule writer to specify how far into a packet Snort should search for a specific pattern.
- *offset* allows the rule writer to specify where to start searching for a pattern within a packet.
- *http_encode* keyword will enable alerting based on encoding type present in a HTTP client request or a HTTP server response.
- *ssl_state* is used to activate the SSL/TLS preprocessor.

Some of these rule options are used in the Experiment Chapter, in this Chapter Snort is used to filter packets. None of the non-payload or post-detection rule options are used, and therefor not mentioned in this chapter.

Chapter 4
Filtering and Bypassing

This chapter will describe how each layer of the OSI could be filtered and some methods on how these filtering mechanisms may bypassed. The filtering methods are based on the threat model in Figure 1.1. The OSI model layers are shown in Figure 4.1

4.1 Physical Layer

The Physical Layer is the lowest layer in the OSI model. Its protocols accepts frames from the Data Link Layer and generate signals in the Network Interface Controller (NIC). There are multiple different materials to send these signalx through, it may be copper, fiber, air, etc. The functionality of the Physical Layer is to detect and accept signals and pass it to the Data Link Layer.

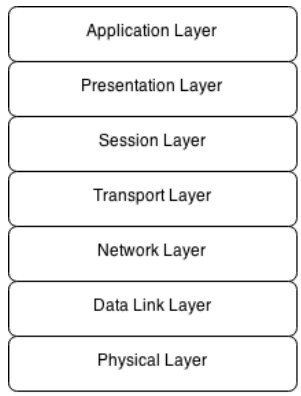

Figure 4.1: The OSI model displayed by layers.

Filter mechanism

The simplest way to filter traffic in the Physical Layer is to disconnected the citizen or group of citizens. This will remove the source of unwanted traffic, and the citizen will not be able to access the content that the government has restricted.

Bypass method

To bypass a physical restriction, the citizen must obtain Internet access through some other medium. This is out of scope for this work.

4.2 Data Link Layer

The Data Link Layer is used to move datagram packets over an individual link. Units of data exchanged by a Data Link Layer protocol are called frames. Almost all Data Link Layer protocols encapsulate each Network Layer datagram packet within a frame before transmitting over a link. The frame consists of a data field, and a number of header fields.

The Data Link Layer offers error detection, re-transmission, flow control, and random access to all frames that are sent and received. Some of the most common protocols in the Data Link Layer are Ethernet, 802.11 wireless Local Area Network (LAN), and Point-to-Point Protocol (PPP).[Kur10]

Filter mechanism

At the Data Link Layer it is possible to filter traffic based on the Media Access Control (MAC) address. A challenge when filtering MAC addresses is that the MAC address is not known to the filterer before unwanted Internet traffic is detected. This makes it possible for the citizen to access content that the government want to restrict. Therefore it is common practice to create a white listing of MAC addresses which is allowed to pass the filter. This is not possible in large scale where new devices are added to the network constantly.

Bypass method

One way to bypass a MAC address filter is to change the MAC address on the filtered device. This is not hard to do, and there are guides on the Internet showing how to do this on various Operating System (OS) and routers. When changing the MAC address on routers it is called MAC address cloning.

4.3 Network Layer

The role of the Network Layer is to move packets from a sending host to a receiving host. This gives it two important features, forwarding and routing. Upon receiving the receiver must check if it is the destination, if that is not the case, packet must be forwarded. The Internets Network Layer provides a service known as best-effort service. The Network Layer is the highest OSI layer that is processed by routers within a network, the layers higher than the Network Layer are implemented in the end system.

One of the protocols of the Network Layer is the IP, it exists in two versions IP version 4 (IPv4) and IP version 6 (IPv6). Each device that wants to communicate with other devices on a network needs and IP address. An anecdote to IP addresses is street addresses; if you want to communicate with another person, you and the other person need a street address to send and receive mail to.

Another protocol is the ICMP, it is the Internets Network Layer error- and information-reporting protocol. The most common way to use ICMP is to use it for error reporting. ICMP messages have a type and a code field. An example of a program that uses the ICMP is the ping program. The program sends an ICMP type 8 code 0 to the specific host, and the host sends back a type 0 code 0 ICMP echo reply. [Kur10]

Filter mechanism

It is possible to filter traffic based on different protocol in the Network Layer. The most common protocols are IP, ICMP and Address Resolution Protocol (ARP). Filtering of some of the protocols in this layer will prohibit network communications, and therefore all communication will disconnected. By filtering IP addresses it is possible to stop traffic from parts or individual citizens in a network.

Bypass method

If protocols which evolves the network are filtered there are no way to bypass this, a network which do not function properly has the same function as a non existing network. Filtering of IP is possible to bypass, this could be done using a VPN or a proxy server. The mechanism to bypass IP filtering depends on where the content is presented. If it is an internal server where a client is filtered as an individual or as a part of a group it is possible to bypass this in two ways. As shown in Figure 4.2, here the blue path uses an internal proxy and the red path uses an external proxy.

If the censored content is not located in the internal network, content may be blocked in two ways. The citizen may be singled out as an individual or group, and

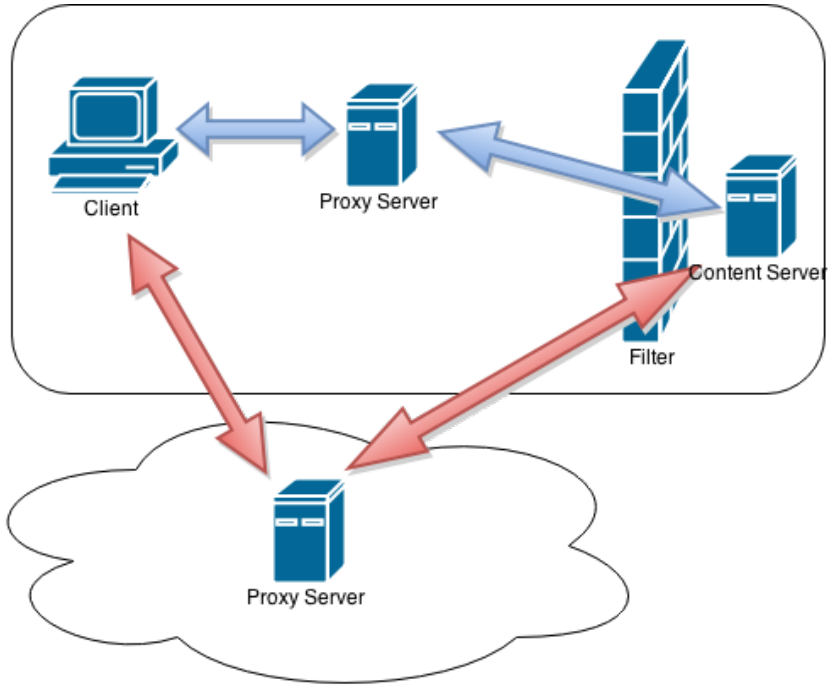

Figure 4.2: Bypassing IP filtering using either an internal or external proxy when accessed service or content is in the home network

their IP address do not have permission to pass the filter. Or the IP address of the server which serve the censored content may blocked. In Figure 4.3 the Filter divides the home and external network, the red path is used when the Clients IP address is blocked, and the blue path is used when the Content Servers IP address is blocked.

When using a Proxy Server, all traffic passes through this server. The Content Server therefore believes that it is the Proxy Server which is requesting the data and not the Client. The Client do not change IP address, it just redirect all its traffic through the Proxy Server.

Another way to bypass IP address filtering when content is outside the internal network is by using a VPN. This will work in the same way as the Proxy Server, but the Client needs to run a VPN client on its machine. This is shown in Figure 4.4. The Client requests a connection to the VPN server and a tunnel is created between the Client and the VPN server. The Filter do not see what is inside this tunnel, this is because the VPN tunnel encrypts all traffic. The Content Server thinks it delivers content to the VPN server and not the Client. The only IP address that crosses the Filter is the Client glsip address and the VPN server IP address.

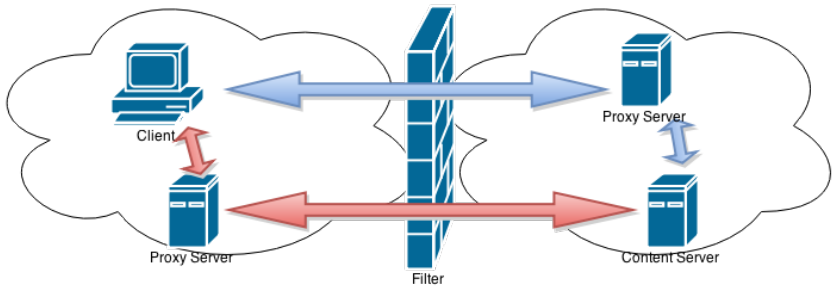

Figure 4.3: Bypassing IP filtering using either an internal or external proxy when accessed service or content is in the external network

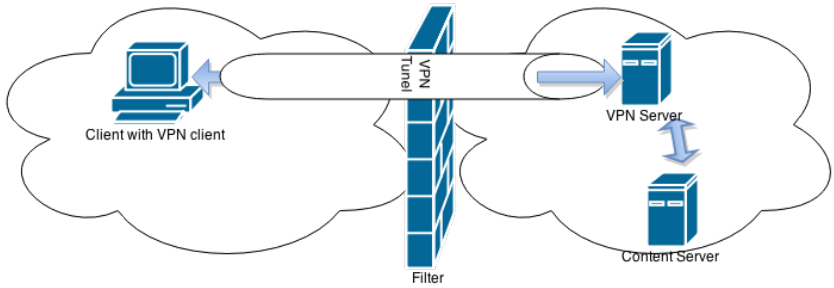

Figure 4.4: Bypassing IP filtering using a VPN server

4.4 Transport Layer

The Transport Layer protocols provides the logical communication between application processes running on different hosts. The protocols in the Transport Layer are implemented in the end systems, not in routers and switches. Two of the best known protocols is TCP and UDP. The UDP provides unreliable, connection-less service to the applications implementing it. TCP provides reliable, connection-oriented service to the application which uses this protocol. [Kur10]

Filter mechanism

At the transport layer it is possible to filter based on protocols, examples of protocols in the Transport Layer are TCP and UDP.

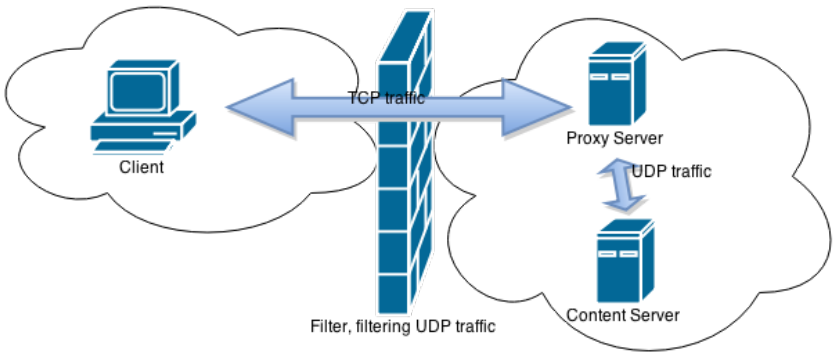

Figure 4.5: Using a proxy to bypass UDP filtering

Bypass method

To be able to serve a service or some content one of the protocols in the Transport Layer must to be able to pass through the filter. If all protocols are filtered this is equal to a disconnected server or network. If the content which the citizen would access is in an external network, the government may set filtering rules to block the protocol that the censored content is served with. To bypass this it is necessary to use a proxy which changes the protocol, in Figure 4.5 the filter blocks UDP traffic, or use a VPN tunnel which uses a TCP connection. In Figure 4.5 the Proxy Server sends and receives traffic from the Content Server using UDP and transmits it back to the Client using a TCP connection, then the content passes through the filter without raising a concern.

4.5 Session Layer

The Session Layer allows devices to establish, manage, and end session. A session is a persistent logical linking of two software application processes, to allow them to exchange data over a prolonged period of time. [Koz05]

Filter mechanism

At the Session Layer there are multiple protocols, the most common protocols are used as support when creating tunnels in the network. The Layer 2 Tunneling Protocol (L2TP) protocol is used to support VPNs, by filtering this protocols it prohibits some VPNs. Another protocol which could be useful to block is the SOCKS protocol, it is used when connecting to a proxy server.

Bypass method

To bypass filtering of the L2TP protocol it is necessary to not use this protocol, if a VPN is needed it is possible to set up a VPN that uses a different protocol.

If the SOCKS protocol is filtered a VPN could be used to get access to a SOCKS proxy. It is also possible to use another type of Proxy Server, such as a HTTP Proxy.

4.6 Presentation Layer

The Presentation Layer has three functions; translation, compression and encryption. The translation function is used when different types of computers are communicating, it helps with translation between the different data representation. Compression and decompression may be used to improve throughput of data. The encryption function ensures the security of the data as it traverse the network, the most common encryption schemes in the presentation layer is the SSL and TLS protocols. [Koz05]

Filter mechanism

Filtering in the Presentation Layer will most likely be by blocking the security protocol s SSL and TLS or the media protocols Joint Photographic Experts Group (JPEG), Moving Picture Experts Group Layer-3 Audio (MP3) and Moving Picture Experts Group (MPEG). Filtering the media protocols will remove pictures and audio which are presented using the protocols mentioned above. By filtering the SSL and TLS protocols, this will remove encrypted traffic flows in the network and protocols such as HTTP Secure (HTTPS) and File Transfer Protocol (FTP) Secure (FTPS) are unsupported. A weakness in the SSL and TLS protocols is that the handshake is not encrypted, this makes it possible to see the source and destination of the encrypted traffic. Because of this weakness, censorship may be conducted on traffic from individuals who are not allowed to access specific content on the Internet.

Bypass method

To bypass the filtering of protocols in the Presentation Layer all the previous methods may be used, proxy and a VPN servers. Changing the different media types would also bypass the filtering of the media protocols.

4.7 Application Layer

All network applications uses the Application Layer, it provides services for user application to employ. A widely used protocol is the HTTP, another is the Simple Mail Transfer Protocol (SMTP) used for e-mail. [Koz05]

Filter mechanism

It is possible to filter traffic based on protocols and data content in the Application Layer. Filtering content which uses the SSL or TLS protocols is not possible, because the data is encrypted. The most used protocol in the Internet is the HTTP.

Bypass method

To bypass protocol filtering it is necessary to use other protocol types, this requires a proxy or VPN server on the other side of the filter. If the filter is filtering on data content, the a bypassing method would be to encode or encrypt the data content.

Chapter 5
Experiment

The purpose of this experiment is to see if it is possible to bypass Snort rules with high likelihood. at the beginning of this chapter, several architectural designs that could be used in the experiment will be proposed. Then some software which is used will be described. Experiment procedures and results are divided in a common sections for each layer in the OSI model.

5.1 Architectural Design

In this section some architectural design proposals that could be used in the experiment will be proposed and discussed.

5.1.1 Alternative 1

The first architectural alternative is shown in Figure 5.1, a Client requesting contents from a Content Server, where the Client and the Content Server are located in separate networks, a router is used to connect the two networks. The Router will be running a filtering mechanism to stop unwanted traffic.

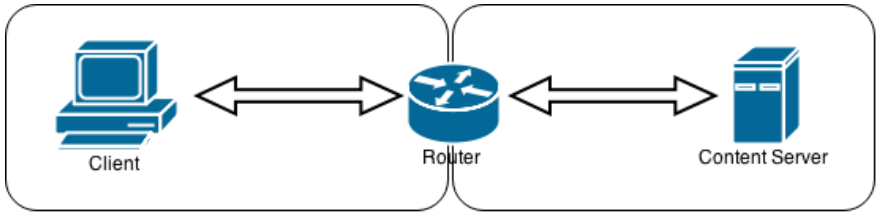

Figure 5.1: Client and Content Server in separate networks, the networks are connected with a router which runs a filtering mechanisms

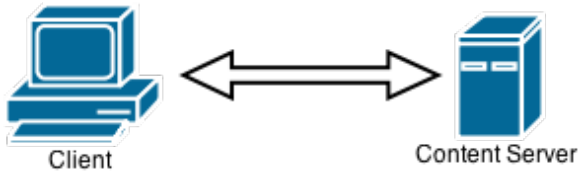

Figure 5.2: Client and Content Server in same network, where the Content Server runs filtering software

Benefits

The network architecture is close to real life architecture.

Restrictions

This alternative needs a router which run a filtering mechanism. This requires extensive configuration or expensive hardware.

5.1.2 Alternative 2

In architectural design alternative two the Client accesses the Content Server directly without any intermediate connections. In this alternative the Content Server will run a filtering software, discarding unwanted traffic itself. This architecture is shown in Figure 5.2

Benefits

This alternative has an easy setup. It can run in any internal network or virtual environment, and do not require additional servers, proxies or routers.

Restrictions

This architecture do net represent the Internet architecture.

5.1.3 Alternative 3

Client requests content from a Content Server, where all traffic passes through a Proxy Server. This alternative is shown in Figure 5.3. The Proxy Server will run the filtering mechanism.

Benefits

This alternative is possible to create in a virtual environment.

Figure 5.3: Client requests content from a Content Server through a Proxy Server

Figure 5.4: Client requesting content from the Cloud through a Proxy Server

Restrictions

Bypassing methods in the Network Layer and Transport Layer uses a Proxy Server. It is not possible for the Client to connect to two different Proxy Server at the same time. So this means that traffic would not go through the Proxy Server which run the filtering software.

5.1.4 Alternative 4

Client doing requests to the Cloud, where all traffic too and from the Client will pass through a Proxy Server. As in Alternative 3 the Proxy Server will run the filtering mechanisms. This alternative is shown in Figure 5.4.

Benefits

This architecture gives an enormous possibility for destinations and applications to access.

Restrictions

The restrictions in this architecture is the same as the restrictions in Alternative 3. Additional there will not be any control of the content of the Internet.

5.2 Setup

5.2.1 Architectural Decision

Out of the four architectural designs two of them will be used. The two alternatives is Alternative 2 and Alternative 3. Alternative 2 will be used when doing experiments in the Network Layer, the Presentation Layer and the Application Layer. This is because of the simple setup. The Alternative 3 will be used in the Transport Layer.

5.2.2 Software Choices

This section describes which software that is used in the Experiment, and why they were selected. The software used in the experiment is:

- VirtualBox
- Ubuntu Server 14.04.1 Long Term Support (LTS)
- Apache Web-Server version 2.4.7
- OpenSSL 1.0.1f
- Snort 2.9.6.0
- Squid 3.3.8
- Burp Suite Free Edition v1.6

VirtualBox

The Content Server in Figure 5.2 and Figure 5.3 is created as a Virtual Network (VM) using VirtualBox. It is setup with a bridged network adapter. This Content Server will run Ubuntu Server, Apache Web-Server and OpenSSL, in Figure Client and Content Server in same network, where the Content Server runs filtering software it will also run Snort.

The Proxy Server in Figure 5.3 is also created as a VM using VirtualBox. The Proxy Server will run Ubuntu Server, Squid and Snort. It is setup with two network adapters, they are configured as listed below:

- Adapter 1
  - Host-only Adapter
  - vboxnet1
  - Promiscuous Mode: Allow VMs

- Adapter 2
  - NAT
  - Promiscuous Mode: Allow VMs

Ubuntu Server

The Content Server will be installed with Ubuntu Server 14.04.1 LTS as the OS.

The Proxy Server in Figure 5.3 will also be installed with Ubuntu Server 14.04.1 LTS as the OS. To get the network interface configuration correct the */etc/network/interfaces* needs to be changed as shown in Listing 5.1.

```
# The primary network interfaces

# Internal network configuration
auto eth0
iface eth0 inet static
        address 10.0.1.101
        network 10.0.1.0
        netmask 255.255.255.0
        broadcast 10.0.1.255

# External network configuration (Internet)
auto eth1
iface eth1 inet dhcp
```

Listing 5.1: Changes in /etc/network/interfaces file in the filter

Apache Web-Server

Apache Web-Server is an open-source HTTP server for modern operating systems. [Fou15]

Apache will be installed on the Content Server using the default Ubuntu Server repositories. To have the Apache Server serve HTTPS content some additional installation steps are necessary. The installation and configuration commands are listed in Listing 5.2

```
sudo apt-get install apache2
sudo a2enmod ssl
sudo service apache2 restart
```

Listing 5.2: Installation of Apache with the SSL module activated

Then a certificate has to be created, this is described in the OpenSSL section. To setup the Apache Web-Server to use HTTPS some further configurations are

needed. The SSL configuration has to be changed, this configuration file is in the */etc/apache2/sites-available/* folder and is called *default-ssl*. Edit this file and add the lines listed in Listing 5.3, where the *IP or Domain_Name* is replaced with either the IP address that the server will serve content from or the Domain Name of the server.

```
ServerName IP or Domain_Name:443

SSLEngine on
SSLCertificateFile /etc/apache2/ssl/apache.crt
SSLCertificateKeyFile /etc/apache2/ssl/apache.key
```

Listing 5.3: Configurations of the default-ssl file in the 5.2.2

The only thing that is remaining is to activate the new Virtual Host, this is done by executing the commands listed in Listing 5.4

```
sudo a2ensite default-ssl
sudo service apache2 reload
```

Listing 5.4: Activate new Virtual Host and restart the 5.2.2

OpenSSL

OpenSSL is an open-source toolkit to implement the SSL and TLS protocols. [Pro15a]

OpenSSL is pre-installed in the Ubuntu Server. There are no necessary configurations to the OpenSSL software. The OpenSSL software is used to create a self signed certificates to the Apache web server. This is done by executing the command listed in Listing 5.5. A precondition for this to work is that there exists a folder inside the */etc/apache2/* directory that is named *ssl*.

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
        -keyout /etc/apache2/ssl/apache.key \
        -out /etc/apache2/ssl/apache.crt
```

Listing 5.5: Creating self signed certificate and key using OpenSSL

Now the Apache Web-Server should be up and running, this may be verified by accessing the IP address of the Content Server on port 80 and 443 through a web browser. A default web page will be displayed, served over HTTP and HTTPS.

Snort

Snort is presented in Chapter 3.

Snort will be setup in alert mode, so that it is possible to look at access logs on the server, instead of using the drop mode where packets are discarded. Snort will be installed from the default Ubuntu Server repository. This is done by executing the command listed in Listing 5.6.

```
sudo apt-get install snort
```

Listing 5.6: Installation of 5.2.2 from the Ubuntu Server repository

To verify that Snort is working the command listed in Listing 5.7 is executed, where *IP* is the IP-range of the network that the Content Server is located on. This will display all network traffic within the IP-range that it detects, in the terminal.

```
snort -d -A console -h IP
```

Listing 5.7: Snort start up commando

Squid

Squid is a caching proxy.[sc13] The Squid software could be used to protect hosts, by setting up the software to act as a firewall and proxying internal traffic.[Wes04]

Squid will be installed on the Proxy Server shown in Figure 5.3. The Squid software is installed from the Ubuntu Server repository by running the command listed in Listing 5.8. Then the configuration file is edited by adding the following lines in the */etc/squid3/squid.conf*.

- acl localnet src 10.0.1.0/24
- http_access allow localnet

The line which contains the *http_port 3128* must be changed to *http_port 3128 transparent*, this is to setup the Squid as a transparent proxy.

```
sudo apt-get install squid3
```

Listing 5.8: Installation of Squid from the Ubuntu Server repository

To load the the changes in the configuration of the Squid software, the Squid software must be restarted. This is done using the command listed in Listing 5.9.

```
sudo service squid3 restart
```

Listing 5.9: Restart command for the Squid software

Burp Suite

Burp Suite is a tool for performing security testing of web applications. It comes with various tools, but only the Proxy tool will be used. [Ltd15]

Burp Suite will be installed on the Client to gain more control of the packets sent to the Content Server. The installation process is minimal, this is because the Burp Suite comes as a Java Archive (JAR) file and is a portable application. This means that as long as Java is installed on the machine it could be executed. Burp Suite is setup to capture and hold packets between the web browser and the NIC of the Client. This makes it possible to modify the packets before they are sent to the network.

5.3 Results

This section contains five experiment with their procedures and results. Throughout the experiments Snort is started with 5 options. These options are *-d*, *-A fast*, *-h IP address*, *-l /var/log/snort* and *-c /etc/snort/snort.conf*.

5.3.1 Packet Capture Capabilities

To check if the Snort software is capable of detecting and alerting packets that passes through the NIC of the Content Server a stress test is conducted. This will show whether Snort is capable of detecting all packets or if some passes through without being analyzed.

Procedure This experiment will be conducted using a Snort rule that collects all traffic that arrives to the Content Server. 9999 packets will be sent to the Content Server within a short period of time, and this will be repeated 100 times to get representative data to analyze.

Result The first rule that were written was the one in Listing 5.10, this was a simple rule that looked for IP packets which originated from the internal network of Norwegian University of Science and Technology (NTNU). This rule created a lot of alerts that did not originate from the Client in Figure 5.4, and therefor it was changed to the one in Listing 5.11. This rule raises an alert for any IP packet which comes from the IP of the Client, and with destination as the IP address of the Content Server.

```
alert ip 129.241.208/23 any -> any any
 (msg:"IP packets from Client to the
  Content Server detected"; sid:10000)
```

Listing 5.10: Snort rule for filtering IP packets with source from internal network and any IP as destination

```
alert ip 129.241.209.185 any -> 129.241.209.152 any
 (msg:"IP packets from Client to the
  Content Server detected"; sid:10000)
```

Listing 5.11: Snort rule for filtering IP packets with Client as the source and Content Server as destination

By doing a Ping of Death from the Client to the Content Server with 9999 packets, it is possible to see how many of these packets Snort do register. The Ping of Death command is shown in Listing 5.12. This command were executed 100 times without Snort active and a 100 times with Snort active.

```
sudo ping -c 9999 -l 65550 129.241.209.152
```

Listing 5.12: Ping of Death command

The number of packets lost is almost identical, this implies that Snort does not affect the throughput to the Content Server. The graphical representation of this in Figure 5.6, shows that the highest number of packets lost were when Snort was not active. When looking at the numbers in Table 5.1 it looks like Snort actually do slow down traffic to and from the server. The RTT min, max and avg are all lower both on average and median. This is also the case when looking at the graphical representation of the average RTT in Figure 5.5. The red points in the figure is average RTT when Snort is active, and the black points are the average RTT when Snort is not running on the Content Server. The red points are visually higher up in the graph than the black points, which indicates that Snort increase the latency of the packets.

Next thing to look at is if Snort raises warnings of all packets sent to the Content Server. The Client and the Content Server are on the same Ethernet network, with no intermediate routers as shown in Listing 5.13. This is to reduce the probability for that packets being dropped in router queues, and other traffic restrictions.

```
traceroute to 129.241.209.152 (129.241.209.152), 64 hops max, \
       52 byte packets
 1  dhcp209-152.ed.ntnu.no (129.241.209.152)  \
       1.393 ms  0.753 ms  0.417 ms
```

Listing 5.13: Route between Client and Content Server, using traceroute

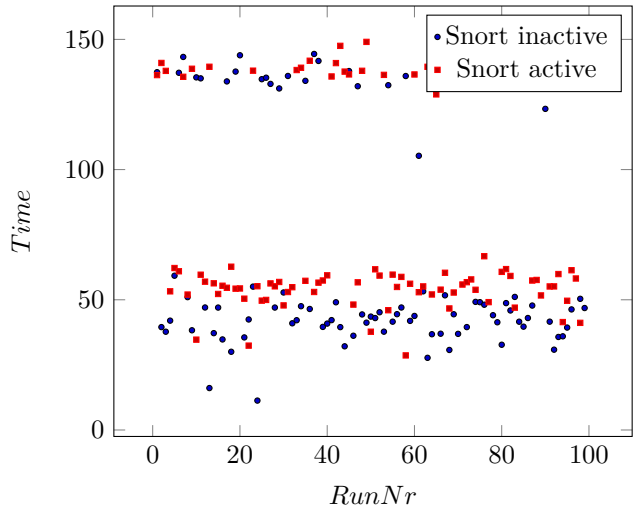

Figure 5.5: Average RTT of the packets which were not lost to the Content Server when testing the throughput.

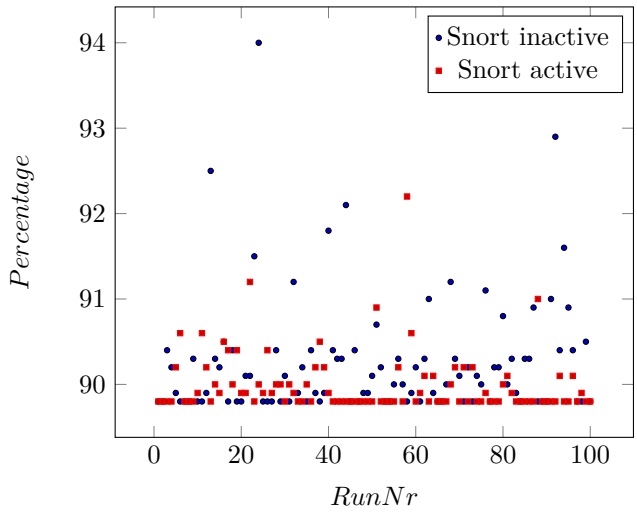

Figure 5.6: Percentage of packets lost of the 9999 packets sent to the Content Server.

Description	Packet loss	RTT min	RTT avg	RTT max	RTT stddev
Snort inactive					
Average	90.266	0.862	69.702	86.542	20.229
Median	90.0	0.842	46.471	59.595	14.382
Snort active with options mentioned in 5.3.3					
Average	89.994	0.877	79.906	104.301	22.579
Median	89.8	0.856	57.409	71.593	17.655

Table 5.1: The average and median packet loss and RTT of traffic to Content Server, when executing the Ping of Death command 100 times, RTT is in ms

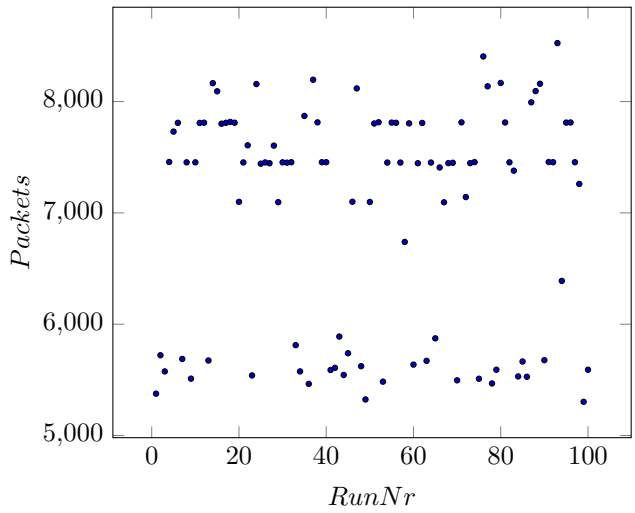

Figure 5.7: Packets registered by Snort for each run.

Snort alerted a total of 699504 IP packets which were sent from the IP of the Client to the IP of the Content Server. The average packet loss of the packets to and from the Content Server were approximate 90%. The percentage of packets which did travel back and forth is thus 10%, this makes $999900 \times 0.10 = 99990$ packets. Snort raised a warning for $\frac{699504}{999900} \times 100 = 69.95739$ of the packets. In Appendix A the script which were used to execute the ping of death are shown. There it is shown that the script have a sleep period of two seconds before it executes the next ping request. This makes it possible to see how many packets Snort registered and alerted for each of the hundred runs. The individual runs with the number of packets registered are shown in Figure 5.7.

Lowest	Average	Highest
5303	6995	8525

Table 5.2: Packets registered by Snort for each run

In Table 5.2 the lowest and the highest number of packets registered by Snort are shown, additional the average number of packets is listed. Based on the numbers in Table 5.2 it is most likely that Snort actually registers all packets that are entering the NIC of the Content Server, and that the packets are lost earlier. When Snort registers a number of packets as high as 8525 packets out of 9999 this is $\frac{8525}{9999} \times 100 = 85.26\%$, with a packet capture $\frac{5303}{9999} \times 100 = 53.04\%$ as the lowest and an average packet capture detection of $\frac{6995}{9999} \times 100 = 69.96\%$. The number of packets missing from a perfect capture at the highest capture run is only $9999 - 8528 = 1471$, and with a total of $9999 - 5303 = 4696$ missing packets at the worst, the missing packets from the lowest up to the highest covers the missing packets; $4696 - 1471 = 3225$, $3225 > 1471$.

When Snort is capable of capture all packets passing through it, and when taking into account that it is an Signature-based Detection IDS, all talk about high likelihood in bypassing a filter is excess. If the packet matches the rule that is loaded into Snort it will be analyzed and an alert will be raised.

5.3.2 Physical Layer and Data Link Layer

There will not be any filtering in the Physical Layer and the Data Link Layer. A physical filter would restrict the Client of any connection to the server, this is not wanted and would not give any results. In the Data Link Layer it is possible to filter on MAC addresses, when the Clients are new to a network the MAC address is not known. Thus they are not in the filtering rules and are not filtered. Additional the Snort software do not support MAC address filtering.

5.3.3 Network Layer

In the Network Layer layer it is possible to filter based on multiple protocols, but Snort do only support IP address filtering and ICMP filtering. This restricts the filtering done in the Network Layer to only include IP addresses and ICMP. In the experiment all ICMP packets, and traffic from IP addresses that originates from the same network as the server will be filtered.

The Snort rules that are used in this experiment is shown in Listing 5.14 and Listing 5.15. Here Snort will be filtering on IP addresses within the range of 129.241.208.0/23, and the protocol type ICMP.

```
alert ip 129.241.208.0/23 any -> 129.241.209.152 any
 (msg:"IP packet from Internal network detected"; sid:10000)
```

Listing 5.14: Snort rule used to alert IP packets which originates from Internal network

```
alert icmp any any -> 129.241.209.152 any
 (msg:"ICMP packet detected"; sid:10001)
```

Listing 5.15: Snort rule used to alert ICMP packets

Procedure First to verify that the rule is alerting when it should, five ping request are executed from a Client which are on the Internal network. This should trigger both rules. The ping request is shown in Listing 5.16. Then the Content of the Server will be accessed using a web browser, this will generate a TCP connection to the Server and should only trigger the rule in Listing 5.14. When the Client has jumped to another network, by using a proxy or moving physically, the command in Listing 5.16 is executed once more. This should only trigger the rule in Listing 5.15. After the ping command result is registered the content of the Content Server are accessed once more via a web browser, and this should not raise any alerts. This experiment is shown in Figure 5.8.

Result In the first run where the ping command in Listing 5.16 are executed from the Client when it is on the same network as the Content Server, Snort do alert with both *IP packet from Internal network detected* and *ICMP packet detected*. When the Content Server is accessed using a web browser, Snort only alerts *IP packet form Internal network detected* as expected. After the Client has moved to an External network the ping command is once more executed. Snort raises five alerts: *"ICMP packets detected"* from the Client. When accessing the Content Server with a web browser Snort do not raise any alerts. This is the result which were expected, when the Client moves to another network it changes its IP address, in this case the IP address of the Client is *78.91.65.8*. Snort do only check for IP addresses within the range of 129.241.208.0 to 129.241.209.255.

An alternative way to bypass IP address filtering is by using an HyperText Markup Language (HTML) iframe as a browser. Then the Client will be viewing content on another Server than the one which are not filtered, and this Server will request content from the Content Server via its iframe.

```
ping -c 5 129.241.209.152
```

Listing 5.16: Ping command executed in the Network Layer experiment

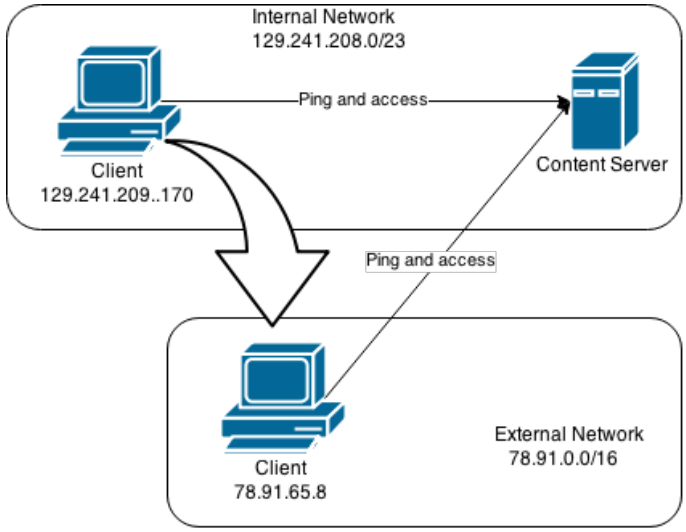

Figure 5.8: Illustration of Client moving from Internal Network to External Network to bypass IP filtering.

To be able to bypass the Snort rule in Listing 5.15 the protocol can not be ICMP. The traceroute tool used in the Packet Capture Capabilities Chapter uses the UDP protocol to send packets. The ping command is a way to check connectivity. By using traceroute as shown in Listing 5.17 the connectivity is also checked, but now running over UDP instead of ICMP. A benefit with this tool is that all the intermediate routers are shown as well. When using traceroute instead of ping to check connectivity to the Content Server, Snort do not raise any alert. This means that the ICMP rule is bypassed.

```
traceroute 129.241.209.152
```

Listing 5.17: Using traceroute to check connectivity and bypassing ICMP filtering

5.3.4 Transport Layer

When entering the Transport Layer there are multiple protocols to filter on, but also here the Snort software limits the number of protocols. Snort only supports the TCP and UDP protocol filtering. The experiment will therefor only include filtering of one of these protocols.

The Snort rules which are used in the Transport Layer experiments are listed in Listing 5.18. In this experiment the Snort rule will filter data packets that runs over

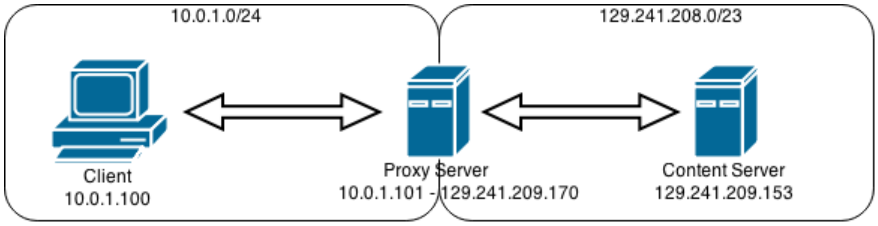

Figure 5.9: The network overview of the setup for experiment on the Transport Layer

TCP. In this experiment the architecture presented in 5.1.3 is used. The environment overview with network IP addresses are shown in Figure 5.9

```
alert tcp any any -> 129.241.209.152 any
 (msg:"TCP packet detected"; sid:10010)
```

Listing 5.18: Snort rule to be used in Transport Layer experiment

Procedure Using a Client that connects to the internet through a proxy, were the proxy is running Snort with the rule listed in Listing 5.18. The Client will try to retrieve content from the Content Server via a web browser and this should raise a TCP alert. When this is verified the Client will connect to a VPN server using a VPN plugin in the browser. This VPN plugin will send traffic over UDP to the VPN server, and then the VPN requests content from the Content Server using TCP. This should not raise any alerts and the Client should be able to gain access to the content of the Content Server.

Result To verify that the Snort rule works, the Client requests the web content on the Content Server. This did not generate an alert as it should have done. The reason was that the Client was setup to use the Proxy Server, and the packets that was sent through the Proxy did not have the IP address of the Content Server as the destination address. Therefor the Snort rule had to be modified, instead of filtering on the destination, the source were filtered instead. The new rule is shown in Listing 5.19.

```
alert tcp 10.0.1.100 any -> any any
 (msg:"TCP packet detected"; sid:10010)
```

Listing 5.19: Snort rule used in Transport Layer experiment

When the rule was changed the expected results were obtained. Then another unexpected error occurred, it is not possible to connect to a VPN if the Client uses a Proxy to gain Internet access. This is because of the Proxy software used, Squid do

only support protocols in OSI layer seven and up, the VPN requires access to OSI layer three to establish a connection.

In theory a VPN connection using UDP will not raise an alert when passing through the Proxy Server, but this was not verified.

5.3.5 Session Layer

Filtering in the Session Layer will not be a part of this experiment. To filter packets in the Session Layer it is necessary to filter packets based on their contents which match with the contents of the different protocols headers in the Session Layer. There are seven different protocols, none of them are commonly used to access web servers. And content matching will be covered in the Application Layer experiment.

5.3.6 Presentation Layer

In the Presentation Layer layer there is one protocol which will be filtered, it is the TLS protocol. The TLS protocol offers encryption for the most common protocols in the Application Layer layer and encrypts the content of them.

The experiment in the Presentation Layer will consist of filtering of the TLS protocol, where it is used to encrypt HTTP traffic. Snort has a preprocessor called SSLPP, it analyzes SSL and TCP traffic and optional determines if and when Snort should stop inspection of it. By enabling the SSLPP and the noinspect_encryption options, only the SSL handshake of each connection will be inspected. The SSLPP is activated by writing rules that uses it. One of the rules used in this experiment uses this and is listed in Listing 5.20. In addition the rule listed in Listing 5.21 will be applied to the Snort software.

```
alert tcp any any -> 129.241.209.152 any
 (msg:"HTTPS client hello detected"; ssl_state:client_hello; sid:10021)
```

Listing 5.20: Snort rule used to enable the SSLPP

```
alert tcp any any -> 129.241.209.152 443
 (msg:"HTTPS traffic detected"; sid:10020)
```

Listing 5.21: Simple Snort rule used to filter TLS traffic

Procedure To verify that the Snort rules are working, content from the Content Server are to be accessed without using any bypassing methods. This should raise an alert from the SSLPP enabled rule and the other rule in Listing 5.21. Then the HTTPS traffic will be moved to an expected TLS traffic port, which is listed in the Snort preprocessor SSL list. This should not raise an alert from the rule

in Listing 5.21, but it should raise a warning from the rule listed in Listing 5.20. Then the traffic will be moved to a port number which is not listed in the Snort preprocessor SSL list, this should also raise an alert.

Result Snort do alert the HTTPS traffic as expected. It do not raise an alert when the Content Server is accessed using HTTP, but raises an alert from both rules when accessing using HTTPS. When moving the TLS connection to port 4433 instead of the default port 443, and making sure that the 4433 port is listed in the *preprocessor ssl: ports* bulk. The expected result is obtained, now only the rule listed in Listing 5.20 gives a warning of "HTTPS client hello detected". When changing the port to something that is not in the *preprocessor ssl: ports* bulk, choosing 4333, the result is not as expected. The Snort software do not alert of any TLS traffic to the Content Server. The Snort rule is set to listen on any port number, and there is a TLS handshake setup. This implies that the preprocessor overrules the ports that will be inspected.

5.3.7 Application Layer

In the Application Layer, filtering will be done on HTTP traffic. Here the content of packets will be inspected and filtered.

```
alert tcp any any -> 129.241.209.152 any
 (msg:"index.html accessed"; sid:10030;
  content:"index";)
```

Listing 5.22: Snort rule to detect if the index.html is accessed

Procedure First, to understanding how Snort detects content of packets, a filter that raises an alert when the *index.html* page is accessed is used. This rule is listed in Listing 5.22. The Content Server will be accessed using a web browser in the Client, using the IP address as the URL. Then adding a trailing */index.html* to the URL. When the rule alerts a specific URL, the */index.html* will be URL-encoded to see if Snort understands this.

Then the same process will be executed against another rule, this rule will look at the packets rawbytes. This rule is listed in Listing 5.23. Here the *69 6E 64 65 78* is the hexadecimal representation of *index*.

```
alert tcp any any -> 129.241.209.152 any
 (msg:"index.html accessed, rawbytes"; sid:10031;
 content:"|69 6E 64 65 78|"; rawbytes;)
```

Listing 5.23: Snort rule viewing rawbytes to detect access to index.html

Result Snort do not raise an alert when accessing the Content Server without the trailing */index.html*, but it do when this is added. When looking in the packet that is sent to the Content Server using Burp Suite, we see what the Client sends. The packet sent from the Client to the Content Server is shown in Listing 5.24. In this request the */index.html* was appended to the URL. When requesting the Content Server without the extended URL the request looks like the on in Listing 5.25

```
GET /index.html HTTP/1.1
Host: 129.241.209.152
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:32.0)
        Gecko/20100101 Firefox/32.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;
        q=0.8
Accept-Language: nb-no,nb;q=0.9,no-no;q=0.8,no;q=0.6,nn-no;
        q=0.5,nn;q=0.4,en-us;q=0.3,en;q=0.1
Accept-Encoding: gzip, deflate
Connection: keep-alive
If-Modified-Since: Wed, 28 Jan 2015 11:15:56 GMT
If-None-Match: "250-50db47ecb69f7-gzip"
Cache-Control: max-age=0
```

Listing 5.24: Request to Content Server capture, using Burp Suite. Here with the /index.html added to the URL

```
GET / HTTP/1.1
Host: 129.241.209.152
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:32.0)
        Gecko/20100101 Firefox/32.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;
        q=0.8
Accept-Language: nb-no,nb;q=0.9,no-no;q=0.8,no;q=0.6,nn-no;
        q=0.5,nn;q=0.4,en-us;q=0.3,en;q=0.1
Accept-Encoding: gzip, deflate
Connection: keep-alive
If-Modified-Since: Wed, 28 Jan 2015 11:15:56 GMT
If-None-Match: "250-50db47ecb69f7-gzip"
```

Listing 5.25: Request to Content Server capture, using Burp Suite

It is the first line of the two requests which is of interest, Snort does seem to only inspect what actually is in this requests. In Listing 5.24 the word *index* is not present, thus this packet do not raise an alert.

Now the *index.html* will be URL-encoded, the encoding of *index.html* is shown in Listing 5.26. When inspecting the request packet using Burp Suite, the GET request has changed. Instead of the *index.html* it is replaced with the text in Listing 5.26. This packet do not raise an alert when passing through the Snort rule in Listing 5.22, but it do load the web page on the Content Server.

```
/%69%6E%64%65%78%2E%68%74%6D%6C
```

Listing 5.26: URL-encoding of index.html

When testing with the Snort rule in Listing 5.23 the same results are obtained. Snort do not raise an alert when accessing the URL without the trailing *index.html* path, nor when URL-encoding *index.html* to the string in Listing 5.26. The rule is verified to be working, when accessing the Content Server with */index.html* appended to the IP address it do raise an alert.

5.4 Conclusion

Snort do not affect the throughput to a Server, but it do increases traffic latency. This indicates that the Snort do not allow packets through itself before the packets are inspected. This is wanted behavior when trying to block access of a service or censored content.

The rules written in Snort is possible to bypass. Using underlying network protocols or by changing the layout of the traffic. An interesting finding was that the SSL and TLS preprocessor overruled the port listed in the Snort rule header. The rule header implies that is should detect traffic on any ports, but it do only analyze traffic on ports which are listed in the preprocessor ssl: ports list.

Chapter 6
Known Bypassing Solutions

In this chapter two solutions which cover some areas of how to avoid censorship will be presented. One of the solutions is developed by Brandon Wiley a PhD student at University of Texas at Austin, and the other is from the Tor Project. Brandon Wiley has started a project which he calls Operator [Wil15], this is a RSS news reader which uses the Dust engine to bypass filtering mechanisms. To use the Dust protocol it requires two entities, a client and a server which understands the protocol.

The Tor Project has a project which they call Pluggable Transport, this project is mostly to prohibit censoring of Tor traffic flows.

6.1 Dust

Dust is a protocol designed to protect against censoring mechanism which uses DPI to fingerprint protocols for the purpose of blocking or rate limiting connections. The protocol creates packets which consists entirely of encrypted or random single-use bytes, this is to be indistinguishable from each other and random packets. The Dust protocol do not anonymize the sender or receiver, this apply to both the IP address and port. All Dust packets can be chained inside either a TCP or UDP packet, making the protocol agile and applicable. [Wil11]

6.1.1 Protocol

Dust uses a two way handshake. The first message is an initial out-of-band invite packet, it contains the server's IP address, port number, public key, the id, and the secret. This invite message is encrypted with a password, this makes it indistinguishable from random bytes. The invite message and the password is then sent to a peer, using communication channels such as email and instant messaging. The second message, which completes the handshake, is a intro packet. The peer uses the IP address and port number from the invite packet to send the intro packet back to the server, this packet is encrypted using the secret sent in the invite packet and

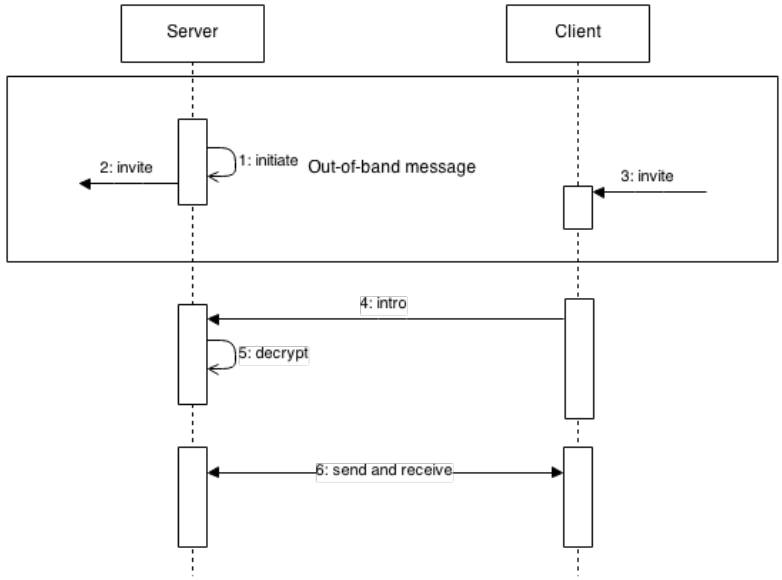

Figure 6.1: Dust Protocol showing the invite, intro and data packet in a message sequence diagram

contains the public key of the peer. The server then decrypts the intro packet with its initial secret, and then generates a shared session key and ads this key to the known hosts list. This two way handshake is shown in Figure 6.1. The server and peer are now able to send and receive encrypted data packets freely.

6.2 Pluggable Transport

The mission of the Tor Project is to be the global resource for technology, advocacy, research and education on the ongoing pursuit of freedom of speech, privacy rights online, and censorship circumvention. They have been widely concerned about anonymity, but lately they have experienced censorship of their traffic. This have resulted in a new product which they call Pluggable Transport, the Pluggable Transport avoids censorship by changing tor traffic flows to look like nothing or something which is expected. A simple representation of a Pluggable Transport proxy called *obfsproxy* is shown in Figure 6.2. Censors who monitor traffic between a Tor Client and a Tor Bridge will now see innocent-looking transformed traffic instead of Tor traffic.

The *obfsproxy* supports transformation of Tor traffic to look like regular traffic, it supports multiple protocols one of them being HTTP. This proxy is one out of

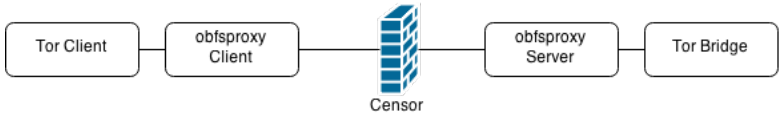

Figure 6.2: Representation of the Pluggable Transport proxy called obfsproxy

ten mentioned on the Tor: Pluggable Transports page[Pro15b], where also Snort is listed.

Chapter 7
Competition

This work were originally meant to be supported by giving participants of a competitions the possibility to act as a censoring authority, but after much struggling with configuration of both Snort and Squid for the experiment, and pages of configuration pages read. The decision to not go through with the competition and trial was made. It would have taken too long for the participants to get familiar with the software, and the challenges with scoring the participants are also difficult.

The intended participants were students at NTNU ranging from first to fifth class students. There are no tuition of IDS at NTNU, and this is not something which individuals are likely to do on their own.

Before the decision of not complete the competition was made, alternative competition setups was proposed. The rest of this Chapter will be a description of the proposed architectures.

In Figure 7.1 a simple network layout of the competition is shown. Where the participants should restrict traffic passing the Proxy Server that connects the two networks. The participants gain points when an unwanted flow of data is stopped, and lose points if they stop all or benign data.

7.1 Architectural Design

In this section some architectural design proposals that could have be used in the competition proposed and discussed.

7.1.1 Alternative 1

The first proposal is shown in Figure 7.2, having one client which request content from a server. Here all requests goes through the participants proxy server. This

Figure 7.1: Network layout of the simulated ISP experiment, where the Proxy Server acts as a gateway between the two networks

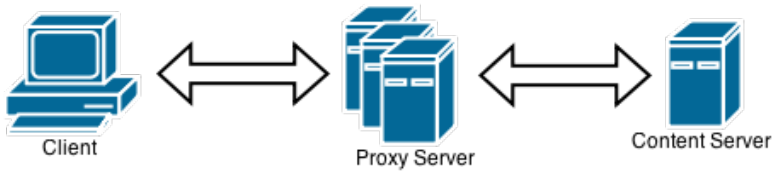

Figure 7.2: One client running requests through multiple proxies to a content server

requires the Client to have some software that connects to each proxy, and requests multiple sources from the Content Server.

This solution makes it possible to score the participants in two ways. The response from the Content Server could be checked on the Client, and requests from the Client could be checked on the Content Server. A restriction to this architecture is that the filtering would only be possible in OSI layer seven and eight.

7.1.2 Alternative 2

To be able to filter traffic in the lower network layers, the clients can not set the proxy settings in software. In this case it is necessary to duplicate clients in a one to one ratio to the proxy servers. This is illustrated in Figure 7.3, here the number of clients mach the number of proxy servers. This alternative is also possible to do with two teams playing against each other, where one of the teams controls the proxy filters, and the other team will try to bypass the filters.

After doing the experiment this architecture, if using the same software as in the experiment (Squid), would not solve the problem with the possibility to only filter in two highest network layers.

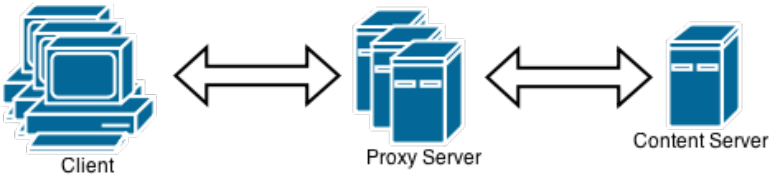

Figure 7.3: Number of client matches the number of proxy servers

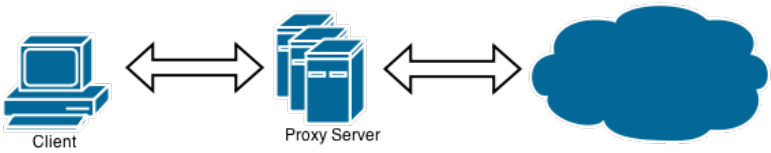

Figure 7.4: One client making request to the Internet through the participants proxy servers

7.1.3 Alternative 3

The previous alternatives have a Content Server, this alternative is a variant of 7.1.1 without the Content Server. Here the Client requests content from the Internet. This would remove the work of creating the content of the Content Server, and possible increase the complexity of the traffic flowing through the Proxy Server. An illustration is shown in Figure 7.4.

Chapter 8
Discussion

In the Experiment chapter it was shown that Snort did analyze all the packets which passed the NIC of the computer where Snort was running. This means that all traffic is analyzed and it is not possible to bypass the Snort software if it has a rule that matches the content of the packet. This makes it impossible to circumvent filters by using on statistics, all traffic is analyzed or detained until resources are available, thus the packet will either circumvent the filter or not. So to bypass an IDS such as Snort it is necessary to "fool" the rules of the tool. In the Experiment all experiments were conducted with knowledge of how the rules, and what the rules were looking for in the network traffic. This is likely not the case in real life, when considering the threat model in this work, where a citizen of a country is censored by the government. This makes it more difficult to change the traffic behavior, and it might not be as easy to discover if traffic is filtered or if something else is the root to the problem. Another problem is that the citizens who is being censored probably is not technical skilled, and most likely not aware of the different methods of circumvent censorship. However within a network as large as a country network, it would be difficult for the government to have knowledge of all the computers in the network. This would make it possible to setup a server inside the network without the government noticing it. This server could be a vpn server, proxy server, or one of the solutions mentioned in Chapter 6.

In the Experiment all the rules were bypassed in some way, except the Transport Layer experiment. Bypassing of the rules in the Transport Layer experiment works in theory, and would most likely have been successful t in practice as well. This shows that it is possible to bypass a DPI if the right method is used. All layers could be bypassed using a bypassing method that is effective in a lower layer. As shown in Figure 4.1, the application layer is higher up in the OSI model than the network layer. This means that if some data in the application layer is filtered, then a bypass method in in the network layer would have the same affect as if the rule in the application layer were bypassed.

Is it possible to use Dust or Pluggable Transport to hide malicious attacks? When malicious contents are delivered, they need to pass into an internal network and a host. In the threat model for this scenario the internal network is a company, where the attacker is in the Internet. To be able to deliver the code to the company network, the attacker has to fool a system or individual to request the malicious code and download it. This is not impossible, but the system or individual must use either the Dust or Pluggable Transport software. If this is the case it is possible to use Dust and Pluggable Transport software to masquerade malicious code to bypass NIDS. This would be difficult for a system administrator to detect, because the purpose of these solutions is to look like normal benign Internet traffic. The conclusion must be to not use such software when located in a sensitive environmental.

A problem with Signature-based Detection IDS such as Snort is that they rely on signatures and rules from databases. These rules and signatures may be written by a community or by a system administrator, a restriction of this is that new hacks or malicious content and behavior will not be detected. If an attacker has a malicious code and it is detected and put in a signature database, then the attacker could just change some lines of code, or payload content and the fingerprint of the malicious code has changed. This would cause the rules of the Signature-based Detection IDS to be noneffective, the "new" malicious code would pass the IDS without raising an alert.

Chapter 9
Future Work

Internet censorship is important and awareness of it should be spread.

A way to spread the word and enlighten students would be to complete the competition, but change the perspective. Creating a competition where the participants were censored and will try to bypass Internet censorship.

By supporting the solutions mentioned in Chapter 6 and other similar solution, Internet censorship might be a word which only existing in history books.

References

[Ber] Matthew Berge. Intrusion detection faq: What is intrusion detection? http://www.sans.org/security-resources/idfaq/what_is_id.php. Accessed: 2015-02-01.

[Cis14] Cisco. Snort users manual 2.9.7. http://manual.snort.org/, 2014. Accessed: 2015-02-01.

[FH13] Freedom-House. Freedom on the net. https://freedomhouse.org/report/freedom-net/freedom-net-2013, 2013. Accessed: 2015-02-01.

[Fou15] The Apache Software Foundation. Apache http server project. http://httpd.apache.org/, 2015. Accessed: 2015-02-01.

[Koz05] Charles M Kozierok. *The TCP/IP guide: a comprehensive, illustrated Internet protocols reference*. No Starch Press, 2005.

[Kur10] Ross Kurose. *Computer networking: a top-down approach*. Pearson Addison-Wesley, 2010.

[Ltd15] PortSwigger Ltd. Burp suite. http://portswigger.net/burp/, 2015. Accessed: 2015-02-01.

[Mah03] Matthew V Mahoney. Network traffic anomaly detection based on packet bytes. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 346–350. ACM, 2003.

[Pro15a] OpenSSL Project. About the openssl project. http://openssl.org/about/, 2015. Accessed: 2015-02-01.

[Pro15b] The Tor Project. Tor: Pluggable transport. https://www.torproject.org/docs/pluggable-transports.html.en, 2015. Accessed: 2015-02-01.

[R+99] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999.

[Reh03] Rafeeq Ur Rehman. *Intrusion detection systems with Snort: advanced IDS techniques using Snort, Apache, MySQL, PHP, and ACID*. Prentice Hall Professional, 2003.

[sc13] squid cache.org. Squid: Optimising web delivery. http://www.squid-cache.org/, 2013. Accessed: 2015-02-01.

[UT14] USA-Today. Top 10 internet-censored countries. http://www.usatoday.com/story/news/world/2014/02/05/top-ten-internet-censors/5222385/, note=Accessed: 2015-02-01, February 2014.

[Wes04] Duane Wessels. *Squid: The Definitive Guide: The Definitive Guide*. " O'Reilly Media, Inc.", 2004.

[Wil11] Brandon Wiley. Dust: A blocking-resistant internet transport protocol. *Technical report. http://blanu.net/Dust.pdf*, 2011.

[Wil15] Brandon Wiley. Operator. http://operatorrss.org/, 2015. Accessed: 2015-02-01.

Appendix A

ping.py

```
import os, time

def executePing():
        for x in xrange(1,101):
                filename = "run_%s.txt" %(x)
                os.system("sudo ping -c 9999 -l 65550 129.241.209.152 > "
                                        + filename)
                time.sleep(2)

def carveResults():
        results = ""
        for x in xrange(1,101):
                results += "------RUN %i------\n" %(x)
                filename = "run_%s.txt" %(x)
                with open(filename) as f:
                        for line in f:
                                if 'packets' in line:
                                        results += line
                                elif 'round-trip' in line:
                                        results += line
                f.close()
        f = open('results.txt','w+')
        f.write(results)
        f.close()

if __name__ == '__main__':
        executePing()
        carveResults()
```