NTNU
Norwegian University of
Science and Technology

Visualization of large scale Netflow data

Nicolai H Eeg-Larsen

Master of Science in Communication Technology
Submission date: June 2016
Supervisor: Otto Wittner, ITEM
Co-supervisor: Arne Øslebø, ITEM

Norwegian University of Science and Technology
Department of Telematics

Title: Visualization of large scale Netflow data

Student: Nicolai Eeg-Larsen

Problem description:

UNINETT is the national research IP network operator in Norway. UNINETT provides universities, university colleges and research institutions with access to the global internet as well as access to a range of online services. UNINETT also offers counselling and acts as secretary and coordinator in collaborative activities between the institutions interconnected by UNINETT. Netflow data provides the network operators, like UNINETT, with detailed information about how the network is being used. It is possible to look at things like who is generating the most traffic, who is the target of DoS attacks, who is sending out spam etc. Netflow is also commonly used for automatic anomaly detection. The challenge here is that the amount of Netflow data collected in a typical backbone network is so large that even the best of anomaly detection algorithms fails by either not detecting the anomalies or by providing too many false positives. Another alternative approach to processing Netflow data with anomaly algorithms is to use visual analytics, i.e. present large amounts of data as pictures and animations such that humans may detect anomalies by visual inspection. This project's main objective will be to survey existing methods and tools for doing advanced visualization of Netflow data, and suggest and test a novel combination of these on data from the UNINETT network.

Responsible professor: Otto J. Wittner, ITEM

Supervisor: Arne Øslebø, UNINETT

Abstract

Networks are forwarding more and more data every year, thus its harder to reveal malicious traffic among all of it. Current systems might not be capable of keeping up with the rapid pace of the growth, and might not be able to separate ill-willed from harmless traffic, or point out to many false positives and not be of much use.

Cisco´s NetFlow standard is used to monitor networks, and due to its versatility it can be used to reveal several interesting things about a network, such as Distributed Denial of Service (DDoS)-attacks or port scans.

In this work I will compare the current solution up against a visual solution developed for this thesis. Carefully choosing different visual elements to represent NetFlow data as clear as possible using the D3.js framework. Using nfdump it is possible to extract specific information and export it into files readable by D3.js code.

The solution was tested with large amounts of data to test if it served its purpose. It was possible to see clear patterns in the data showing the network behavior was repeating itself. It was able to point out abnormalities where single Internet Protocol (IP)-addresses suddenly receives atypical amounts of traffic either distributed across thousands of ports or just to one single one. Since the data is anonymized it is hard to point out what the reason for the peaks in traffic, but shows the solution serves its purpose to separate the irregular from the harmless traffic.

Through further testing it was concluded that the solution has great purpose, but cannot serve as a stand alone solution due to its limitations for going into very specific details while still being intuitive and easy to use. But its functionality complements a command-line-based solution very well by removing a lot of the resource and time consuming commands that would be done in the command-line.

Sammendrag

Nettverk behandler stadig mer og mer data hvert år, og dermed er det vanskeligere å avsløre ondsinnet trafikk i blant denne. Nåværende systemer er ikke i stand til å holde tritt med den raske veksten, og er nødvendigvis ikke i stand til å skille ondsinnet fra ufarlig trafikk, eller vil peke ut for mange falske positiver og ikke være til nytte.

Cisco sin NetFlow standard brukes til å overvåke nettverk, og på grunn av sin allsidighet kan den brukes til å avsløre flere interessante ting om et nettverk, for eksempel DDoS-angrep eller port skanning.

I denne oppgaven vil jeg sammenligne dagens løsning opp mot en visuell løsning utviklet spesifikt for denne oppgaven. Jeg har nøye valgt visuelle elementer til å illustrere NetFlow data så klart som mulig ved å bruke D3.js rammeverket. Ved å bruke nfdump er det mulig å hente ut spesifikk informasjon og eksportere den til filer som kan leses av D3.js kode.

Løsningen ble testet med store mengder data for å teste om det tjente sin hensikt. Det var mulig å se tydelige mønstre i dataen som viser at oppførselen til nettverket gjentar seg. Den var i stand til å peke ut anomalier hvor en enkelt Internet Protocol (IP) -adresse plutselig får unormale mengder trafikk enten fordelt på tusenvis av porter eller bare til en eneste en. Siden dataene er anonymiserte er det vanskelig å peke ut hva årsaken til avvikene i trafikken stammer fra, men viser at løsningen tjener sin hensikt ved å skille uregelmessig fra harmløs trafikk.

Gjennom videre testing ble det konkludert med at løsningen tjener sin hensikt, men kan ikke fungere som en frittstående løsning på grunn av begrensninger for å gå inn i svært spesifikke detaljer, samtidig som den er intuitiv og lett å bruke. Funksjonaliteten komplementerer en tekst basert løsning veldig godt ved å fjerne mange tid og ressurskrevende oppgaver fra kommando-linjen.

Preface

This master's thesis is submitted to the Norwegian University of Science and Technology (NTNU) as the final report in the course TTM4905, and in the final part of a five-year Master of Science in Communication Technology program at the Department of Telematics (ITEM).

I would like to thank Professor Yuming Jiang for giving me the opportunity to write this thesis.

Secondly I want to thank my supervisors at UNINETT, Arne Øslebø and Otto J. Wittner for being a valuable part of the process and helping me along the way with everything from the structure of the thesis to helping with the development of the proposed solution.

Additionally I would like to thank my fellow students sharing an office with me for providing immediate support and being great sparring partners in discussing the challenges I faced.

Promise me you'll always remember: You're braver than you believe, and stronger than you seem, and smarter than you think.

by A. A. Milne

Contents

List of Figures

Listings

List of Tables

List of Acronyms

API Application Programming Interface.

AS Autonomnous Systems.

AVC Application Visibility and Control.

BGP Border Gateway Protocol.

CSS Cascading Style Sheets.

CSV Comma Separated Value.

DDoS Distributed Denial of Service.

DoS Denial of Service.

DPA Data Presentation Architecture.

GB Gigabyte.

HTML HyperText Markup Language.

IETF Internet Engineering Task Force.

IP Internet Protocol.

IPFIX IP Flow Information eXport.

IPv4 Internet Protocol version 4.

IPv6 Internet Protocol version 6.

ITEM Department of Telematics.

NTNU Norwegian University of Science and Technology.

SVG Scalable Vector Graphics.

TPC Transmission Control Protocol.

XML Extensible Markup Language.

Chapter 1
Introduction

1.1 Motivation

Network security, monitoring and Big Data are becoming major pillars in the network industry. Data traffic is growing at an alarming rate[7], and is not showing any signs of slowing down. A majority of this traffic is harmless and not in the interest for network monitoring systems. As the amount of traffic increases, separating malicious from another activity is becoming more difficult. Today network monitoring can be achieved trough using Cisco's NetFlow standard along with tools as nfdump. The visual presentation in such tools are limited, and is not very interactive or intuitive.

Theory on how data is best presented visually dates back decades. The term 'visual presentation' is used to refer to the actual presentation of information through a visible medium such as text or images [8]. It ranges from simple text to body-language, and a good visual communication design is evaluated based on the comprehension by the audience.

Using visualization to represent NetFlow data has not been common, and was first published as a solution after the 2004 ACM Workshop on Visualization and data mining for computer security [2]. In this paper NVisionIP, a java based solution, is proposed. Capable of providing the system state, and possibly with further work, find patterns or attacks on the network.

1.2 Scope and Objectives

1.2.1 Scope

This thesis will cover the basics in how NetFlow data is generated and collected, and also the tools used to manipulate the raw data. As well as showing theory behind general visualization and how to present data such as NetFlow. D3.js is used to create examples using real NetFlow data provided by UNINETT.

Connecting the basics mentioned earlier in this section to a simple solution using real anonymous data and allowing volunteers, with experience using nfdump and existing tools, to provide feedback(see 1.2.2).

1.2.2 Objectives

O.1: Processing anonymous data

UNINETT provided one month of data, which is hundreds of Gigabyte (GB). A large part of the process is analysing and perform queries using nfdump to be able to extract the exact information needed, as well as sorting it and creating files readable by the D3.js framework.

O.2: Developing working solution

Objective two concerns the development of a simple visual solution using D3.js. One of the primary objectives is to use as much of the theory covered in 2.2 effectively.

O.3 Evaluate solution

In objective three the main objective is discussing if the solution created in O.2 display enough potential to be able to use visualization as a tool in network monitoring.

1.3 Outline

- **Chapter 2** covers the basic knowledge of how the NetFlow files are captured, processed and presented. How the different tools used to extract the requested data, and how it works today. Basic visualization theory is presented, along with more specifics about how to present data in the best way possible. The D3.js framework is presented as well.
- **Chapter 3** describes the related works done in this field before, and how it is used today. It depicts how different attacks would look using the NetFlow packets, and how the different attacks work.
- **Chapter 4** shows how the D3.js framework was used to create a working solution and the choices behind each visual element. It also goes into more detail surrounding how certain information is found using nfdump. This chapter is the novel contribution in this work.
- **Chapter 5** discusses the use of the solution after interviews completed with UNINETT-staff. It incorporates real data into the proposed solution, and display how the data looks, and the findings from this.

- **Chapter 6** presents challenges encountered during the work in this paper and expected obstacles in eventual future development.
- **Chapter 7** includes my concluding remarks along with recommendations for future action.

1.4 Methodology

Approaching the problems and objectives defined in this thesis I went through a process taking decisions to select what framework was to be used in the development, how the solution should look like, and how it should be tested. Using basic knowledge within data visualization I made a working solution with D3.js. In order to test the solution I performed qualitative interviews with experts.

1.5 Deviation from problem description

The problem description states two main objectives; the first one is to survey existing methods, and the second is to suggest and test a novel combination of these in data from the UNINETT network. In my research, I found that there was a lack of modern and recent solutions specially made for NetFlow data, and focused on the second objective in this work. Due to this, I chose to use the D3.js framework and create a visual presentation and test how it could represent NetFlow data in the best way possible.

Chapter 2
Background

2.1 NetFlow

Cisco IOS NetFlow standard creates an environment that includes tools to understand who, what, when, where and how network traffic is flowing. NetFlow makes it easier for administrators to utilize the network optimally. One can determine the source and destination of traffic and use this information to reveal for example DDoS-attacks or spam mail. [9]

2.1.1 How does it work?

Every packet that is forwarded within a router/switch is examined for a set of Internet Protocol (IP) packet attributes. With these attributes, one can determine if the packet is unique, or similar to other packets.

The attributes used by NetFlow are:

- IP source address
- IP destination address
- Source port
- Destination port
- Layer 3 protocol type
- Class of service
- Router/Switch interface

To group packets into a flow, one compares source/destination IP address, source/destination ports, protocol interface, and class of service. Then the packets

and bytes are tallied. This method is scalable because a large amount of network information is condensed into a database of NetFlow information called the NetFlow cache, shown in Figure 2.1.

When the NetFlow cache is created one can use this to understand the network behavior. The different attributes generate different knowledge about a certain network and combined they can paint a detailed picture of how the network is working. For example, the ports show what application is utilizing the traffic, while the tallied packets and bytes show the amount of traffic. [1] [10]

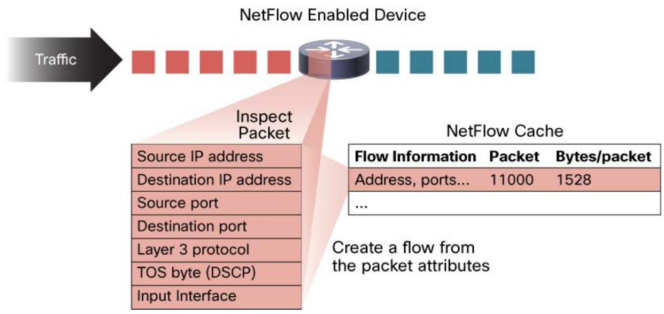

Figure 2.1: Creating a flow in the NetFlow cache [1]

- Source address allows the understanding of who is originating the traffic
- Destination address tells who is receiving the traffic
- Ports characterize the application utilizing the traffic
- Class of service examines the priority of the traffic
- The device interface tells how the network device is utilizing the traffic
- Tallied packets and bytes show the amount of traffic

Additional information added to a flow includes:

- Flow timestamps to understand the life of a flow; timestamps are useful for calculating packets and bytes per second
- Next hop IP addresses including Border Gateway Protocol (BGP) routing Autonomnous Systems (AS)
- Subnet mask for the source and destination addresses to calculate prefixes

– flags to examine Transmission Control Protocol (TPC) handshakes

[1]

2.1.2 Main components

A typical set-up using NetFlow consists of three main components:

– **Flow Exporter:** aggregates packets into flows and exports flow records towards one or more flow collectors.

– **Flow collector:** is responsible for reception, storage and pre-processing of flow data received from a flow exporter.

– **Analysis application:** an application that analyses the received flow data in different contexts, such as intrusion or traffic profiling.

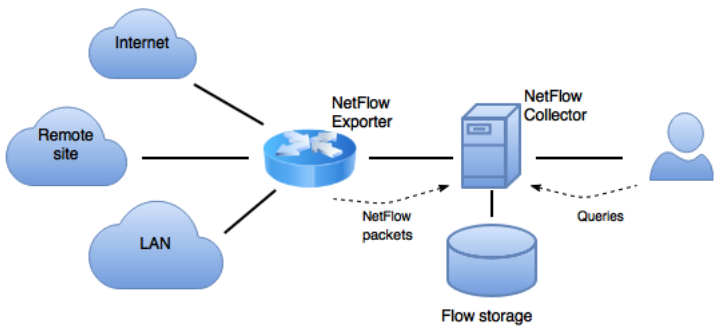

Figure 2.2: Simple NetFlow architecture

2.1.3 nfdump

nfdump collect and process NetFlow data on the command line. It stores NetFlow data in time sliced files. The files are binary, and this provides the possibility of either returning the output from nfdump in the same binary form, or as readable text. nfdump has four output formats, raw, line, long and extended. The challenge of representing Internet Protocol version 6 (IPv6) addresses is handled by shrinking them in the regular output. In Figure 2.3 the collection process is depicted, and in Figure 2.4 the processing of collected NetFlow data is shown.[11]

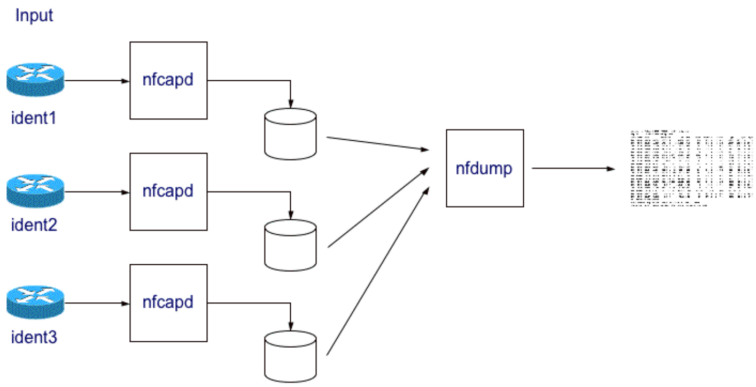

Figure 2.3: NetFlow collection process

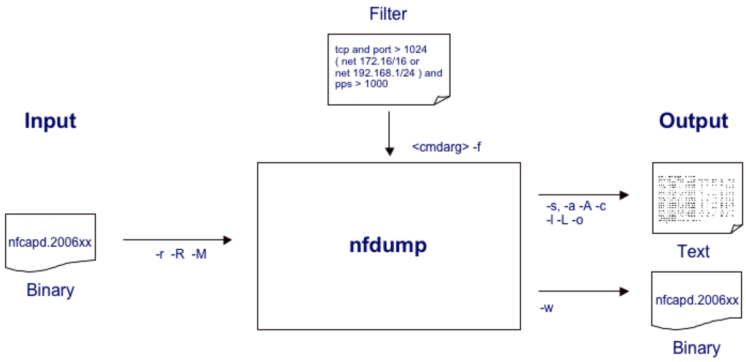

Figure 2.4: Processing of collected NetFlow data

v5,v9 and IPfix

- **v5:** NetFlow v5 is the most popular version of Cisco Netflow. It is fixed, meaning it always stays the same and makes for a simpler deciphering.
- **v9:** v9 is, opposite of its predecessor, dynamic. The collector will need to know the format of incoming NetFlow v9 flows, which means v9 templates periodically needs to be sent to the collector to inform of the format in which the flows are being exported. It was made to support technologies as Multi-cast, IPSec and Multi Protocol Label Switching (MPLS). IPv6 support was added as well. [12]
- **IPFIX:** Based on the design of NetFlow v9, IP Flow Information eXport

(IPFIX) added support for variable length strings, making it possible for Application Visibility and Control (AVC) exports in the future[13]. The IPFIX is a Internet Engineering Task Force (IETF)[14] standard as of RFC 3917[15].

Example of use

An example of a nfdump command used in this project is the extraction of the number of flows each day to find the 10 most used destination IP-addresses:

```
nfdump -R /data/netflow/oslo_gw/2012/01/01/nfcapd.201201010000:nfcapd
    .201201012355 -n 10 -s dstip -o csv > example.csv
```

Listing 2.1: nfdump example

Such a request iterates over many files due to the -R command. In this case, it is all captures between 00:00 until 23:55 on the first of January 2012. It is limited to the 10 most popular destination IP addresses. All of this is stored in a .Comma Separated Value (CSV)-file which is optimal for use with the D3.js framework.

The nfdump-command in 2.1 return the output shown in Figure 2.5.

```
Top 10 Dst IP Addr ordered by flows:
Date first seen          Duration Proto       Dst IP Addr    Flows(%)     Packets(%)       Bytes(%)         pps      bps   bpp
2011-12-31 23:58:52.073 86420.475 any      191.220.233.80   304171( 2.3)   374411( 0.4)  369.4 M( 0.9)        4    34194   986
2011-12-31 23:58:50.706 86421.658 any       162.185.32.85   204693( 1.5)   210870( 0.2)   29.1 M( 0.1)        2     2693   137
2011-12-31 23:58:50.872 86416.387 any      161.220.8.250    171799( 1.3)   175078( 0.2)    7.8 M( 0.0)        2      721    44
2011-12-31 23:58:50.748 86421.701 any      162.185.32.105   171177( 1.3)   367501( 0.4)   21.7 M( 0.1)        4     2009    59
2011-12-31 23:58:50.170 86422.648 any       190.49.180.97   137249( 1.0)   380318( 0.4)  296.5 M( 0.7)        4    27447   779
2011-12-31 23:58:52.698 86419.411 any      161.223.1.106    116698( 0.9)   158786( 0.2)   25.4 M( 0.1)        1     2355   160
2011-12-31 23:58:50.762 86421.892 any      159.152.49.239   114587( 0.9)   696163( 0.8)  953.8 M( 2.3)        8    88288  1370
2011-12-31 23:58:52.421 86419.823 any      190.49.138.164    98869( 0.7)   110728( 0.1)    6.8 M( 0.0)        1      632    61
2011-12-31 23:58:52.881 86419.483 any      190.49.138.168    84728( 0.6)   109403( 0.1)  161.8 M( 0.4)        1    14977  1478
2012-01-01 15:19:13.044 29561.745 any       71.238.74.19     84632( 0.6)   592943( 0.7)  793.9 M( 1.9)       20   214838  1338

Summary: total flows: 13373369, total bytes: 41.8 G, total packets: 86.5 M, avg bps: 3.9 M, avg pps: 1000, avg bpp: 482
Time window: <unknown>
Total flows processed: 13373369, Blocks skipped: 0, Bytes read: 695426912
Sys: 5.373s flows/second: 2488661.8  Wall: 5.407s flows/second: 2473155.2
```

Figure 2.5: Output from the nfdump command in 2.1

NfSen

NfSen is a graphical web-based front end for the nfdump NetFlow tools. It allows the user to navigate through the NetFlow data, and shows a simple graphic of the data. An example of the interface is shown in Figure 2.6.

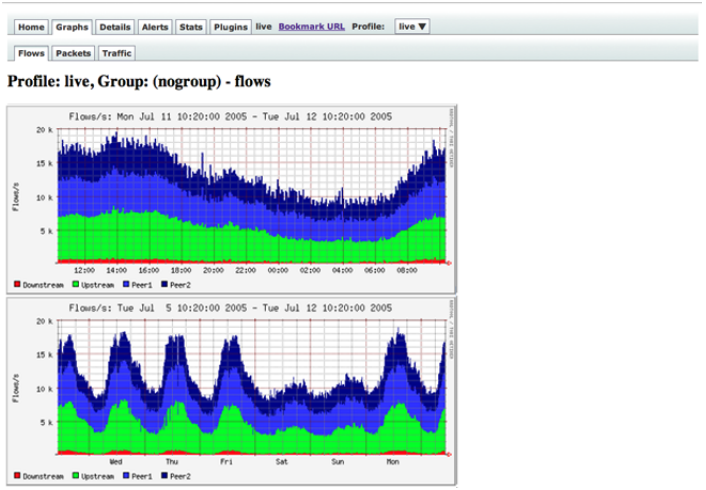

Figure 2.6: Example of the interface in NfSen

2.2 Data visualization

Data visualization refers to the techniques used to communicate data or information by encoding it as visual objects. Meaning that information is represented by any visual element such as graphs and plots, but may also take any other visual form. Visualization helps users analyze and interact with data in a whole new way. It makes complex data more accessible, understandable and usable.[16]

In recent years, the rate of which data is generated has increased rapidly, and the need for information to be available and comprehensible is growing. All these new sources of data have created what we refer to as "Big Data" [17]. Without visual presentation, such data is often too big to understand under human inspection. This is one of the major reasons data visualization is such an emerging technique to interpret Big Data.

Combining several parameters through visualization could reveal something automated systems might ignore or don't pick up.

> The greatest value of a picture is when it forces us to notice what we never expected to see.
>
> John Tukey

2.2.1 Characteristics

In his book from 1983, The Visual Display of Quantitative Information[18], Edward Tufte states that characteristics of an effective graphical representation should:

- show the data
- induce the viewer to think about the substance rather than about methodology, graphic design, the technology of graphic production or something else
- avoid distorting what the data has to say
- present many numbers in a small space
- make large data sets coherent
- encourage the eye to compare different pieces of data
- reveal the data at several levels of detail, from a broad overview to the fine structure
- serve a reasonably clear purpose: description, exploration, tabulation or decoration
- be closely integrated with the statistical and verbal descriptions of a data set.

2.2.2 Visual perception

In this work the correlation between effective visual communication and how it is perceived upon human inspection is important. A human's ability to distinguish between differences in length, shape and color is referred to as "pre-attentive attributes".

A good example of this is imagining finding the number of a certain character in a series of characters. This requires significant time and effort, but if the character were to stand out by being a different size, color or orientation this could be done through pre-attentive processing quickly. Good data visualization takes all of this into consideration and uses pre-attentive processing. In the example, shown in Figure 2.7 and 2.8, it is easy to see how pre-attentive processing is used to distinguish how many occurrences of the number 5 is in a larger set of random numbers.
znaps.net

2.2.3 Data presentation architecture

The purpose with Data Presentation Architecture (DPA) is to identify, locate, manipulate, format and present data in such a way as to optimally communicate meaning and proffer knowledge[19]. This has become an important tool in Business Intelligence, the art of transforming raw data into something useful.

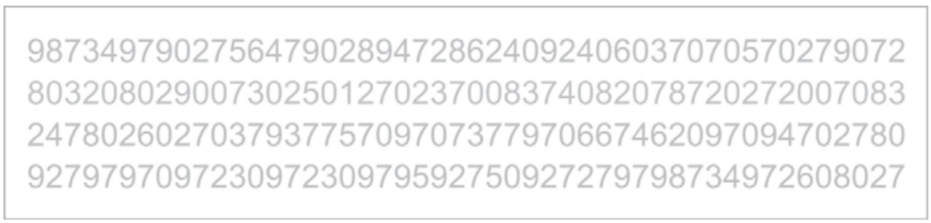

Figure 2.7: Example of dataset of random numbers where no pre-attentive processing is done

Figure 2.8: Example of a dataset of random numbers where pre-attentive processing has been used to distinguish the occurrences of the number five

Objectives

DPA has two main objectives, which is the following:

- To use data to provide knowledge in the most efficient manner possible (minimize noise, complexity, and unnecessary data or detail given each audience's needs and roles)
- To use data to provide knowledge in the most effective manner possible (provide relevant, timely and complete data to each audience member in a clear and understandable manner that conveys important meaning, is actionable and can affect understanding, behaviour and decisions)

Scope

The actual work of DPA consist of:

- Creating effective delivery mechanisms
- Define relevant knowledge needed by each viewer

- Determine how often the data should be updated
- Determine how often and when the user needs to see the data
- Finding the right data
- Utilizing the best visualizations and presentation formats

2.3 D3.js

In this work D3.js [5] is chosen as the framework to create examples of effective data visualizations due to its dynamical and interactive properties. D3 stands for Data-Driven Documents, and is a Javascript library. D3.js allows users to bind arbitrary data to a Document Object Model. It uses widely implemented SVG, Cascading Style Sheets (CSS) and HyperText Markup Language (HTML)5 standards. D3 is unique in the way it creates SVG objects from large datasets using simple D3.js functions to generate rich text/graphic charts and diagrams.

2.3.1 How does it work?

The W3C DOM API, often used in today's web development, is often tiring to use. An example bit of code from [5] shows how one changes the text color of paragraph elements:

```
1
var paragraphs = document.getElementsByTagName("p");
3 for (var i = 0; i < paragraphs.length; i++) {
  var paragraph = paragraphs.item(i);
5   paragraph.style.setProperty("color", "white", null);
}
```

Listing 2.2: HTML example [5]

In D3.js this could be solved trough one line of code:

```
1 d3.selectAll("p").style("color", "white");
```

Listing 2.3: D3.js example [5]

D3.js also possess dynamic properties which gives the user a powerful tool to create advanced graphics with a small amount of code.

This next snippet of code shows how the D3.js framework simply appends to an existing HTML object. [20]. The output from 2.4 can be seen in Figure 2.9.

```
1 <!DOCTYPE html>
<meta charset="utf-8">
```

```
<style> /* set the CSS */

body { font: 12px Arial;}

path {
    stroke: steelblue;
    stroke-width: 2;
    fill: none;
}

.axis path,
.axis line {
    fill: none;
    stroke: grey;
    stroke-width: 1;
    shape-rendering: crispEdges;
}

</style>
<body>

<!-- load the d3.js library -->
<script src="http://d3js.org/d3.v3.min.js"></script>

<script>

// Set the dimensions of the canvas / graph
var margin = {top: 30, right: 20, bottom: 30, left: 50},
    width = 600 - margin.left - margin.right,
    height = 270 - margin.top - margin.bottom;

// Parse the date / time
var parseDate = d3.time.format("%d-%b-%y").parse;

// Set the ranges
var x = d3.time.scale().range([0, width]);
var y = d3.scale.linear().range([height, 0]);

// Define the axes
var xAxis = d3.svg.axis().scale(x)
    .orient("bottom").ticks(5);

var yAxis = d3.svg.axis().scale(y)
    .orient("left").ticks(5);

// Define the line
var valueline = d3.svg.line()
    .x(function(d) { return x(d.date); })
    .y(function(d) { return y(d.close); });

// Adds the svg canvas
var svg = d3.select("body")
```

```
55      .append("svg")
            .attr("width", width + margin.left + margin.right)
57          .attr("height", height + margin.top + margin.bottom)
        .append("g")
59          .attr("transform",
                  "translate(" + margin.left + "," + margin.top + ")");
61
// Get the data
63  d3.csv("data.csv", function(error, data) {
        data.forEach(function(d) {
65          d.date = parseDate(d.date);
            d.close = +d.close;
67      });

69      // Scale the range of the data
        x.domain(d3.extent(data, function(d) { return d.date; }));
71      y.domain([0, d3.max(data, function(d) { return d.close; })]);

73      // Add the valueline path.
        svg.append("path")
75          .attr("class", "line")
            .attr("d", valueline(data));
77
        // Add the X Axis
79      svg.append("g")
            .attr("class", "x axis")
81          .attr("transform", "translate(0," + height + ")")
            .call(xAxis);
83
        // Add the Y Axis
85      svg.append("g")
            .attr("class", "y axis")
87          .call(yAxis);

89  });

91  </script>
    </body>
```

Listing 2.4: Example of use of the D3.js framework [6]

This graph would be appended to the body element of the html and look like this:

In the code in 2.4 the dynamic properties are visible as the x- and y-axis change its parameters based on the input data. Having the possibility to use several different data formats lowers the demand to process data before feeding it to the D3.js code.

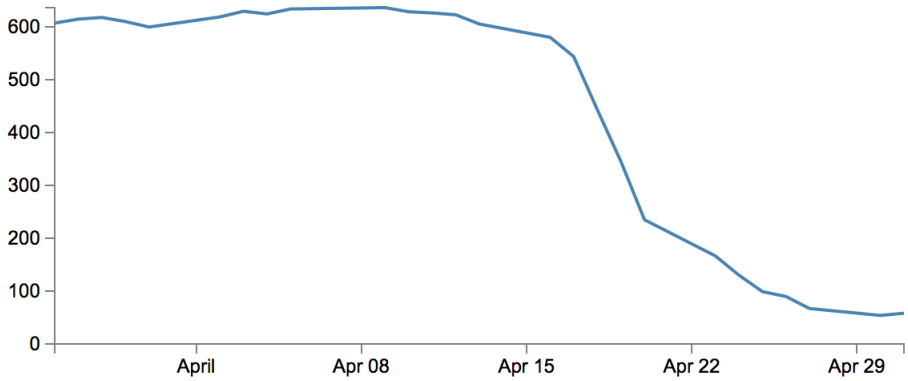

Figure 2.9: Output from the HTML code in 2.4

2.3.2 SVG files

SVG is an Extensible Markup Language (XML)-based vector image format for two-dimensional graphics with support for interactivity and animation. The two latter are favouring attributes of the SVG-format. A great advantage when using SVG figures is that they are defined by text files, which means they can be searched, indexed, scripted and compressed. Being accessible on practically any platform and scaling on any device makes it an extremely versatile solution.

Chapter 3
Research

In this chapter I will cover the research found investigating related works, and how malicious attacks looks from the networks side.

3.1 Related work

In the last decade, the importance of security against attacks on large computer systems has grown rapidly. In 2004, the ACM workshop on Visualization and data mining for computer security presented NVisionIP: netflow visualizations of system state for security situational awareness[2]. This was one of the first tools to visualize NetFlow data. The visualization was based on either number of bytes transmitted or the number of flows to or from the hosts on the network. Due to its limitations, it was not tested further during this work. NVisionIP's interface can be seen in figure 3.1.

In [21] they discuss the use of NVisionIP to combat different security concerns. Most of the same attacks covered in this work are relevant today, although in today's massive amounts of data, they are harder to discover.

- **Worm infection**: One of the most basic security functions one might uncover. Worms usually spread by probing for other hosts. Filtering out hosts transmitting a lot of Flows with a single destination port, one could easily see which machines are infected and should be taken offline.
- **Compromised systems**: If a host is compromised, the attacker might install malware that allows the attacker to control the machine. Following this, an attacker might turn a host into a file server. By detecting large volumes of traffic on certain ports one might discover such an attack.
- **Misuse**: Misuse of computer networks in order with terms of use etc.. An example is detecting if certain users have abnormal high volumes of traffic. By

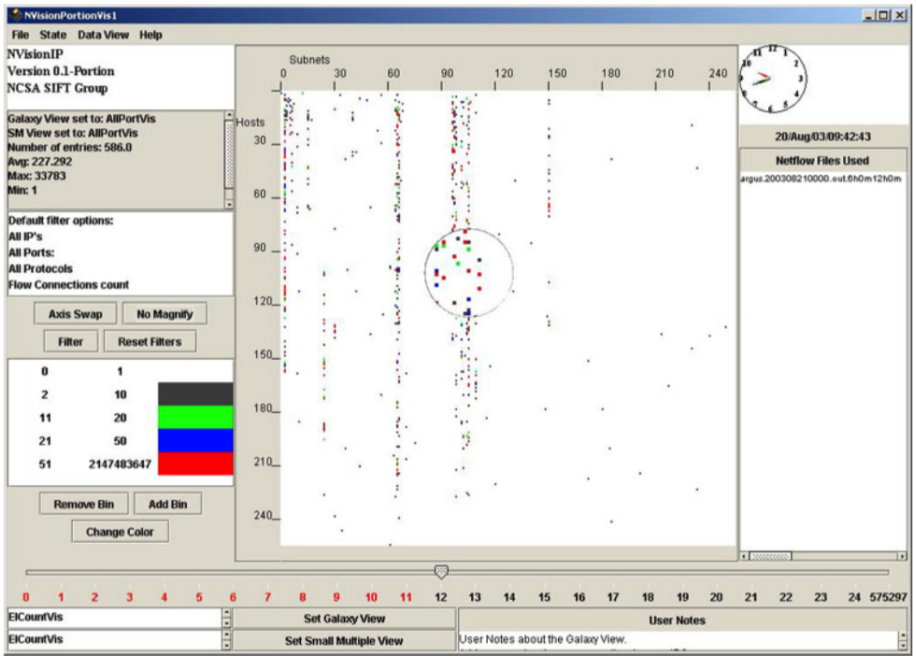

Figure 3.1: Interface in NVisionIP [2]

inspecting further one can uncover if this is through one single application and not in accordance with the policies of the organization.

- **Port Scans**: When a large number of ports are used at a specific host it is easily identified by NVisionIP.
- **Denial of Service (DoS):** Denial of Service Attacks will be visible through spikes in traffic volume from the host attacking. If a host is attacked the same pattern is visible through high volumes in receiving traffic. Thus, peaks in traffic are not necessary an attack, but might be a result of a new release or backup

In [22], a previous masters thesis from NTNU, considers a lot of the same issues covered in this work. It discusses solutions available at the time, and what patterns in the traffic are created when malicious activities takes place.

3.2 Alternatives to D3.js

Along with D3.js, a lot of different frameworks exists that serve the purpose need in this work. chart.js [23] is a popular and widely used framework used to create

graphs. Compared to D3.js it is easier in use, but lacks the possibilities D3.js possess. Due to the broader selection in D3.js it was chosen over its competitors to not limit creativity in the development.

3.3 Initial research

In section 2.1 it was illustrated what the raw format of the NetFlow packets looks like. In this work I will compare different visual representations of large amounts of NetFlow packets up against current command-line based representation. How much more effective is visualization compared to the text format?

To understand this, basic knowledge from visualization will be used to create and test a solution with real NetFlow data. This is where D3.js will come to great use. It can be used to quickly develop simple interactive graphs that can be used to test solutions up against each other.

To be able to identify a DDoS attack, one can look at it from two sides. By finding someone who is attacking, or someone who is being attacked. In this work, we will look at the second scenario. If someone is a target of a DDoS a sudden peak in incoming traffic will appear. Upon further investigation one will look for similar network behavior among previous data to look for a pattern to disclose if there is an actual attack.

3.4 Traits of a DDoS attack

In a DDoS attack, there is a large number of hosts performing the attack. In many cases, a lot of the hosts are not even aware they are a part of an attack. This is called a botnet, derived from the words robot and network. Using compromised systems, called zombies, gives the attacker control of a large enough amount of hosts to perform a volume-based DDoS attack as depicted in figure 3.2.

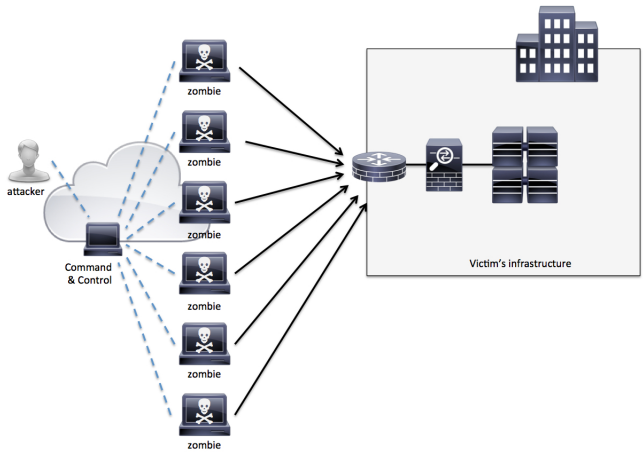

Figure 3.2: Botnet used in a DDoS attack

Another new trend is using large data centres or cloud machines to launch these attacks. Either trough renting or compromising them. As cloud providers are offering such large amounts of computers, this new platform is not only great for legitimate use, but also cyber-criminals. Figure 3.3 shows how an attacker can use cloud-based machines to perform such an attack.

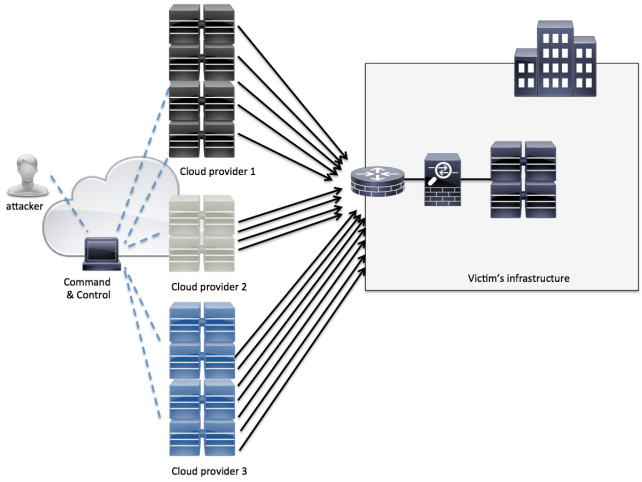

Figure 3.3: Structure of a DDoS attack using cloud providers

Distributed Reflection Denial of Service attacks are becoming more and more frequent. DrDoS techniques usually involve multiple victim host machines that unwillingly participate in a DDoS attack on the attackers primary target. Requests to the victim host machines, are redirected, or reflected, from the victim hosts to the target. One advantage of the DrDoS attack method is anonymity. In a DrDoS attack, the primary target appears to be directly attacked by the victim host servers, not the actual attacker. This approach is called "spoofing" and is illustrated in figure3.4.Amplification is another advantage of the DrDoS attack method. By involving multiple victim servers, the attacker's initial request yields a response that is larger than what was sent, thus increasing the attack bandwidth[3].

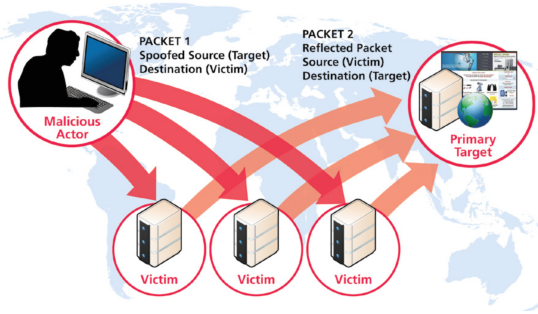

Figure 3.4: How a DrDoS uses victims to attacks their primary target [3]

3.4.1 Raw NetFlow format

```
router#show ip cache flow
IP packet size distribution (90784136 total packets):
  1-32  64  96 128 160 192 224 256 288 320 352 384 416 448 480
  .000 .698 .011 .001 .004 .005 .000 .004 .000 .000 .003 .000 .000 .000 .000
  512 544 576 1024 1536 2048 2560 3072 3584 4096 4608
  .000 .001 .256 .000 .010 .000 .000 .000 .000 .000 .000
IP Flow Switching Cache, 4456704 bytes
 1885 active, 63651 inactive, 59960004 added
 129803821 ager polls, 0 flow alloc failures
 Active flows timeout in 30 minutes
 Inactive flows timeout in 15 seconds
IP Sub Flow Cache, 402056 bytes
 0 active, 16384 inactive, 0 added, 0 added to flow
 0 alloc failures, 0 force free
 1 chunk, 1 chunk added
 last clearing of statistics never
Protocol         Total    Flows   Packets Bytes  Packets Active(Sec) Idle(Sec)
--------         Flows     /Sec     /Flow  /Pkt     /Sec     /Flow     /Flow
TCP-Telnet    11393421      2.8         1    48      3.1       0.0       1.4
TCP-FTP           236       0.0        12    66      0.0       1.8       4.8
TCP-FTPD           21       0.0     13726  1294      0.0      18.4       4.1
TCP-WWW         22282       0.0        21  1020      0.1       4.1       7.3
TCP-X             719       0.0         1    40      0.0       0.0       1.3
TCP-BGP             1       0.0         1    40      0.0       0.0      15.0
TCP-Frag        70399       0.0         1   688      0.0       0.0      22.7
TCP-other     47861004     11.8         1   211     18.9       0.0       1.3
UDP-DNS           582       0.0         4    73      0.0       3.4      15.4
UDP-NTP         287252      0.0         1    76      0.0       0.0      15.5
UDP-other       310347      0.0         2   230      0.1       0.6      15.9
ICMP             11674      0.0         3    61      0.0      19.8      15.5
IPv6INIP            15      0.0         1  1132      0.0       0.0      15.4
GRE                  4      0.0         1    48      0.0       0.0      15.3
Total:        59957957     14.8         1   196     22.5       0.0       1.5
SrcIf         SrcIPaddress    DstIf         DstIPaddress    Pr SrcP DstP  Pkts
Gi0/0         192.168.10.201  Gi0/1         192.168.60.102  06 0984 0050     1
Gi0/0         192.168.11.54   Gi0/1         192.168.60.158  06 0911 0035     3
Gi0/1         192.168.150.60  Gi0/0         10.89.16.226    06 0016 12CA     1
Gi0/0         192.168.10.17   Gi0/1         192.168.60.97   11 0B89 0050     1
Gi0/0         10.88.226.1     Gi0/1         192.168.202.22  11 007B 007B     1
Gi0/0         192.168.12.185  Gi0/1         192.168.60.239  11 0BD7 0050     1
Gi0/0         10.89.16.226    Gi0/1         192.168.150.60  06 12CA 0016     1
router#
```

Figure 3.5: How a DoS attack looks in raw format [4]

In the preceding example, there are multiple flows for UDP port 80 (hex value 0050). In addition, there are also flows for TCP port 53 (hex value 0035) and TCP port 80 (hex value 0050).

The packets in these flows may be spoofed and may indicate an attempt to perform these attacks. It is advisable to compare the flows for TCP port 53 (hex value 0035) and TCP port 80 (hex value 0050) to normal baselines to aid in determining whether an attack is in progress.[4]

Chapter 4
D3.js and NetFlow

In this chapter I will document the development of my solution, and how I process the data to extract what information I want from the captures made available.

4.1 Using D3.js

Earlier in this work I mentioned that D3.js will be used to show examples of effective visualization of NetFlow data. It is assumed that the data has already been processed before it is made accessible. I was supplied with two months of anonymous data from UNINETT to get familiar with NetFlow and be able to use real data for my visualizations. This is anonymized data from January of 2012 from Trondheim and Oslo NetFlow collectors. Data from these collectors are sampled in the ratio of either 1/100 or 1/1000. Sampling may lead to some deviations in the networks behavior, but due to the nature of the attacks investigated in this work, sampling should not be an obstacle. Attacks where only a specific flow is crucial, such a sampling ratio could exclude these packets, and the attack would go by unnoticed.

4.1.1 Scope

Large amounts of data should be presented with such a scope that is intuitive and easy to understand. I have chosen to focus on the most popular destination IP-addresses and ports over different time periods. Going into detail looking at an hour by hour view of each IP and port. I felt this was the best way to represent the data.

IP spectrum

Choosing the address spectrum as the main focus, one will have to find a way to represent the entire IPv4 spectrum. This is alone a challenge, and when it comes to IPv6, it becomes practically impossible due to the range of the addresses. This means one will have to use pre-processing to separate out the IP-addresses worth examining closer. In the data provided by UNINETT, it is possible to e.g., list the

top 10 files in size, meaning the time slots with the most flows. This query provided the results in Figure 4.1 when performed on the provided data.

```
[eeglarse@iou2:/data/netflow$ find . -printf '%s %p\n'|sort -nr|head
14838848 ./oslo_gw/2012/01/18/nfcapd.201201181325
14729440 ./oslo_gw/2012/01/18/nfcapd.201201181335
14729284 ./oslo_gw/2012/01/18/nfcapd.201201181310
14720548 ./oslo_gw/2012/01/18/nfcapd.201201181330
14687944 ./oslo_gw/2012/01/18/nfcapd.201201181315
14651908 ./oslo_gw/2012/01/18/nfcapd.201201181340
14566420 ./oslo_gw/2012/01/18/nfcapd.201201181320
14563196 ./oslo_gw/2012/01/18/nfcapd.201201181305
14508804 ./oslo_gw/2012/01/18/nfcapd.201201181345
14472664 ./oslo_gw/2012/01/18/nfcapd.201201181300
```

Figure 4.1: The files with the highest amount of flows from the provided files

From this simple preprocessing it is easy to see that in the time period between 1300-1400 on the 18th of January there was a clear peak in the number of flows claiming all the spots in the top 10. If we compare them to the files with the lowest amount of flows, shown in Figure 4.2, we see a major span between the files with the highest and lowest numbers of flows.

Through this command we create a .csv file containing the hour in question. With this file it is possible to find the most used destination IP-addresses, shown in Figure 4.3.

One particular address is clearly separated from the others. The high numbers could be a DDoS attack or other abnormalities, but not does not necessary need to be ill-willed. If we look at the list of top IP-addresses sending packets, we see in Figure 4.4 that the same IP-address, 192.239.62.2, occurs here as well.

```
[eeglarse@iou2:/data/netflow$ find . -printf '%s %p\n'|sort -nr|tail -100
849408 ./trd_gw1/2012/01/01/nfcapd.201201010920
848160 ./trd_gw1/2012/01/01/nfcapd.201201010745
842856 ./trd_gw1/2012/01/01/nfcapd.201201010725
834212 ./trd_gw1/2012/01/01/nfcapd.201201010655
832340 ./trd_gw1/2012/01/01/nfcapd.201201010640
830364 ./trd_gw1/2012/01/01/nfcapd.201201010555
828856 ./trd_gw1/2012/01/01/nfcapd.201201010845
821940 ./trd_gw1/2012/01/01/nfcapd.201201010900
816012 ./trd_gw1/2012/01/01/nfcapd.201201010905
804624 ./trd_gw1/2012/01/01/nfcapd.201201010735
802596 ./trd_gw1/2012/01/01/nfcapd.201201010545
799164 ./trd_gw1/2012/01/01/nfcapd.201201010635
780600 ./trd_gw1/2012/01/01/nfcapd.201201010650
772644 ./trd_gw1/2012/01/01/nfcapd.201201010610
760424 ./trd_gw1/2012/01/01/nfcapd.201201010705
758864 ./trd_gw1/2012/01/01/nfcapd.201201010720
```

Figure 4.2: The files with the lowest amount of flows from the provided files

```
eeglarse@iou2:~$ cat test_180112.csv |cut -f 5 -d ',' |sort|uniq -c|sort|tail
  17762 162.185.32.85
  19878 161.222.192.123
  21506 191.220.233.80
  23995 161.223.1.164
  37704 161.223.1.108
  39759 159.152.145.176
  49316 161.223.1.142
  51467 190.49.180.97
  61424 161.223.1.106
 120976 192.239.62.2
```

Figure 4.3: Top ten used destination addresses within the timeframe 1300-1400, 18th of January

```
eeglarse@iou2:~$ cat test_180112.csv |cut -f 4 -d ',' |sort|uniq -c|sort|tail -10
  18502 161.222.192.123
  18557 191.220.233.80
  19338 162.185.32.85
  29367 161.223.1.164
  46376 192.239.62.2
  47139 190.49.180.97
  47844 161.223.1.108
  50509 161.223.1.142
  77527 159.152.145.176
  83184 161.223.1.106
```

Figure 4.4: Top ten source addresses within the timeframe 1300-1400, 18th of January

```
eeglarse@iou2:~$ nfdump -R /data/netflow/oslo_gw/2012/01/18/nfcapd.201201180000:nfcapd.201201182355 -n 10 -s dstport 'dst ip 192.239.62.2'
Top 10 Dst Port ordered by flows:
Date first seen          Duration Proto          Dst Port    Flows(%)      Packets(%)       Bytes(%)          pps      bps   bpp
2012-01-18 08:05:55.302 21426.006 any              19424      224( 0.0)      284( 0.0)    284614( 0.0)        0      106  1002
2012-01-18 09:08:59.910 23270.338 any               8981      216( 0.0)      287( 0.0)    298504( 0.0)        0      102  1040
2012-01-18 08:07:44.051 31093.199 any              51750      207( 0.0)      223( 0.0)    252365( 0.0)        0       64  1131
2012-01-18 09:11:42.207 39490.631 any              40293      156( 0.0)      219( 0.0)    190328( 0.0)        0       38   869
2012-01-18 07:57:48.578 31937.555 any              31019      153( 0.0)      601( 0.0)    838518( 0.0)        0      210  1395
2012-01-18 08:44:26.768 25405.425 any              32586      138( 0.0)      398( 0.0)    534649( 0.0)        0      168  1343
2012-01-18 10:07:20.801 28733.933 any              35708      113( 0.0)      312( 0.0)    419602( 0.0)        0      116  1344
2012-01-18 11:06:40.021 14417.571 any               7211      113( 0.0)      357( 0.0)    489458( 0.0)        0      271  1371
2012-01-18 09:00:24.646 15560.048 any              65267      107( 0.0)      213( 0.0)    292432( 0.0)        0      150  1372
2012-01-18 07:47:39.195 29887.494 any              55562      106( 0.0)      558( 0.0)    770732( 0.0)        0      206  1381

Summary: total flows: 830383, total bytes: 1.9 G, total packets: 1.6 M, avg bps: 178956, avg pps: 18, avg bpp: 1224
Time window: <unknown>
Total flows processed: 44250006, Blocks skipped: 0, Bytes read: 2301034764
Sys: 2.840s flows/second: 15579254.6 Wall: 2.843s flows/second: 15559172.3
```

Figure 4.5: Top 10 most popular destination ports and the amount of flows

To further investigate the activity on one certain IP-addresses, one can look at ports, both source and destination. In the case of the mentioned IP-address the flows are widely spread across thousands of destination ports in Figure 4.6. If we look at the destination ports in Figure 4.7, we note that 90.8% of the traffic is originated from port 80 meaning this is probably someone uploading a large file to our IP-address.

```
eeglarse@iou2:~$ nfdump -R /data/netflow/oslo_gw/2012/01/18/nfcapd.201201180000:nfcapd.201201182355 -n 10 -s dstport 'dst ip 192.239.62.2'
Top 10 Dst Port ordered by flows:
Date first seen          Duration Proto         Dst Port    Flows(%)     Packets(%)       Bytes(%)         pps      bps   bpp
2012-01-18 08:05:55.302 21426.006 any              19424      224( 0.0)      284( 0.0)   284614( 0.0)         0      106  1002
2012-01-18 09:08:59.910 23270.338 any               8981      216( 0.0)      287( 0.0)   298504( 0.0)         0      102  1040
2012-01-18 08:07:44.051 31093.199 any              51750      207( 0.0)      223( 0.0)   252365( 0.0)         0       64  1131
2012-01-18 09:11:42.207 39490.631 any              40293      156( 0.0)      219( 0.0)   190328( 0.0)         0       38   869
2012-01-18 07:57:48.578 31937.555 any              31019      153( 0.0)      601( 0.0)   838518( 0.0)         0      210  1395
2012-01-18 08:44:26.768 25405.425 any              32586      138( 0.0)      398( 0.0)   534649( 0.0)         0      168  1343
2012-01-18 10:07:20.801 28733.933 any              35708      113( 0.0)      312( 0.0)   419602( 0.0)         0      116  1344
2012-01-18 11:06:40.021 14417.571 any               7211      113( 0.0)      357( 0.0)   489458( 0.0)         0      271  1371
2012-01-18 09:00:24.646 15560.048 any              65267      107( 0.0)      213( 0.0)   292432( 0.0)         0      150  1372
2012-01-18 07:47:39.195 29887.494 any              55562      106( 0.0)      558( 0.0)   770732( 0.0)         0      206  1381

Summary: total flows: 830383, total bytes: 1.9 G, total packets: 1.6 M, avg bps: 178956, avg pps: 18, avg bpp: 1224
Time window: <unknown>
Total flows processed: 44250006, Blocks skipped: 0, Bytes read: 2301034764
Sys: 2.840s flows/second: 15579254.6 Wall: 2.843s flows/second: 15559172.3
```

Figure 4.6: Top 10 most popular source ports and the amount of flows

```
eeglarse@iou2:~$ nfdump -R /data/netflow/oslo_gw/2012/01/18/nfcapd.201201180000:nfcapd.201201182355 -n 10 -s srcport 'dst ip 192.23
9.62.2'
Top 10 Src Port ordered by flows:
Date first seen          Duration Proto         Src Port    Flows(%)     Packets(%)       Bytes(%)         pps      bps   bpp
2012-01-17 23:58:26.575 86431.371 any                80   753701(90.8)    1.5 M(93.6)    1.8 G(95.7)        17   171173  1251
2012-01-17 23:59:47.399 86318.714 any               443    70569( 8.5)    86400( 5.5)   67.7 M( 3.5)         1     6269   782
2012-01-18 00:00:04.416 86245.775 any                53     2068( 0.2)     2068( 0.1)   304428( 0.0)         0       28   147
2012-01-18 08:06:13.613 38095.737 any              8080     1283( 0.2)     2158( 0.1)    2.9 M( 0.2)         0      616  1360
2012-01-18 08:16:18.036 37528.850 any              8182      492( 0.1)     5510( 0.3)    7.8 M( 0.4)         0     1666  1418
2012-01-18 08:14:50.332 37657.316 any               364      238( 0.0)      937( 0.1)    1.3 M( 0.1)         0      282  1420
2012-01-18 08:07:44.051 31093.199 any              8088      222( 0.0)      238( 0.0)   256549( 0.0)         0       66  1077
2012-01-18 08:05:55.302 17131.174 any              8004      218( 0.0)      276( 0.0)   274566( 0.0)         0      128   994
2012-01-18 10:51:48.542 15511.827 any              8003      212( 0.0)      281( 0.0)   289984( 0.0)         0      149  1031
2012-01-18 08:51:01.496 26206.793 any              8000      172( 0.0)      231( 0.0)   165360( 0.0)         0       50   715

Summary: total flows: 830383, total bytes: 1.9 G, total packets: 1.6 M, avg bps: 178956, avg pps: 18, avg bpp: 1224
Time window: <unknown>
Total flows processed: 44250006, Blocks skipped: 0, Bytes read: 2301034764
Sys: 2.858s flows/second: 15479569.6 Wall: 2.856s flows/second: 15489030.1
```

Figure 4.7: Top 10 most popular source ports and the amount of flows sent to IP-address 192.239.62.2

Time spectrum

From the Netflow packets, it is also possible to look at the time spectrum. Here one looks at the amounts of flows within one time slot, not separating the different IP-addresses. With the vast amount of IP-addresses this is not a suitable spectrum to present the data to find specific attacks etc.. It could, however, be used to monitor amounts of traffic over time or which ports are in use at certain times to use the bandwidth better, etc..

4.2 Number of flows to a certain host and port

In this section, the process of creating a visual interface, and the reasoning behind the selection of the different visual elements is described.

4.2.1 Scope in D3.js

Three modules of a solution are presented to show different levels of detail. It combines both the time spectrum and IP-spectrum to investigate the NetFlow data. The data from UNINETT required pre-processing before being made available to the D3.js solution. These bash scripts used can be found in Appendix B. The code for the solution can be found in Appendix A.

File structure

With the bash scripts, thousands of files are created. Data is split into as small and many files as possible to reduce loading time at the front-end, and to make sure the data is as comprehensible as possible. In section 2.2.2 we described the importance of pre-processing the data to highlight certain aspects of larger data sets. With these scripts we perform advanced preprocessing on significant amounts of data to bring out the desired output to best find deviations trough visualization.

Overview

First visual element we have is the overview which in this case show an entire year, neatly separated into months, weeks and days, shown in Figure 4.8. The purpose of this is to be able to recognize patterns quickly in the data that correlates to regular activities as backup etc. as mentioned in 3.1. For example a weekly backup will create similar levels of network usage at specific times each week.

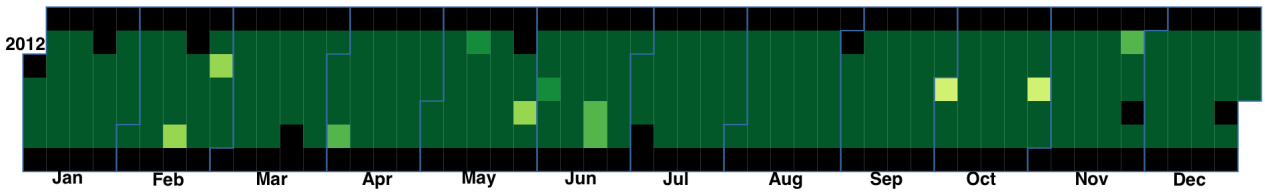

Figure 4.8: Proposed visual element showing statistics for an entire year

IP-addresses and ports

For each day there are millions of different combinations of IP-addresses and ports that send flows between each other. Through pre-processing it is possible to distinguish which IP-addresses are the most popular each day, and thus find the ports the IP-addresses use the most. The example in 4.9 visualize the number of flows for each of these combinations trough a heatmap. A heatmap distinguishes values through a color range based on the highest values in the data set, meaning the higher the value, the darker the color.

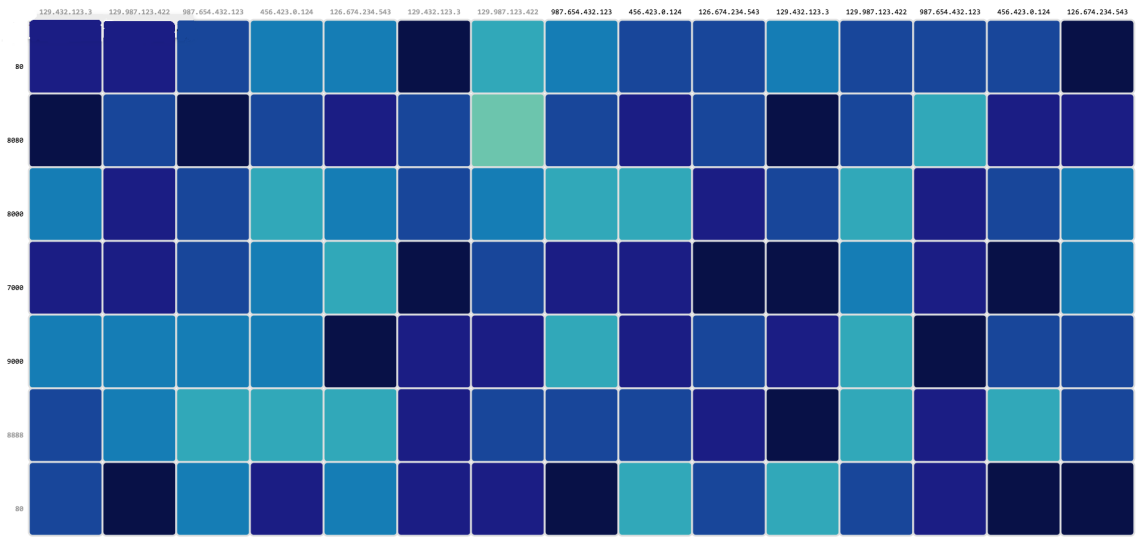

Figure 4.9: Proposed visual element showing statistics for IP and port combinations

24-hour chart

When a IP-address and port are selected for a specific day, the next step is to look at that day in more detail. Using the time-spectrum this graph shows the 24-hour lapse and the amount of flows at each hour for the chosen IP-address and port.

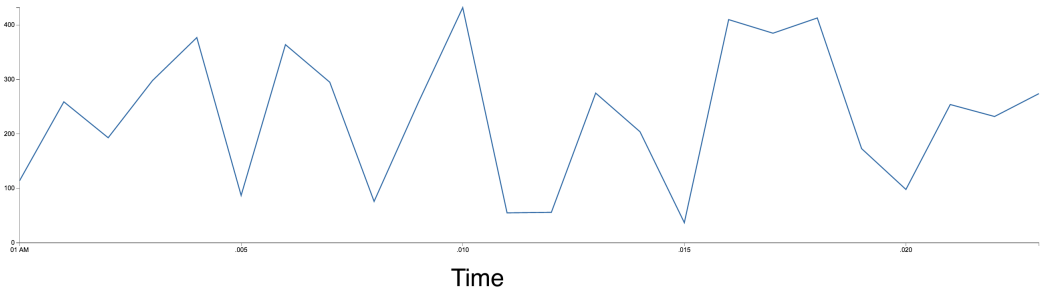

Figure 4.10: 24-hour chart showing the behavior on one single IP and port combination

4.2.2 Pros and cons

In this section, I will present pros and cons for the visual- and text-based solution. Both the solutions serves their purpose in different ways. Some of the cons of each of them are complemented well by the others pros.

Visual solution

Pros	Cons
Very intuitive and straightforward to use: The need for competence within nfdump etc. is not as high, and lowers the threshold to use the tool.	A visual solution might limit the possibilities that exist in an advanced command line based solution. If going into detail on the data, a visual solution might be limited to certain parameters and will not show the bigger picture.
Visual interpretation of data is quicker and easier for the human mind to comprehend. Patterns and other aberrations are more likely to be visible than in a text based solution.	
As mentioned in 2.2.3, a goal is to show only the data relevant to the user. This means that the data that is not pertinent to the user is never made visible.	

Table 4.1: Pros and cons of a visual solution

Command line solution

Pros	Cons
A command line solution possesses a wide range of very specific commands as in nfdump, providing the possibility to do very distinct searches.	Such a tool is time-consuming and harder to master than a intuitive and visual solution. One is very reliable on the searches being correct and providing the exact answer one is looking for requires a fair amount of skill. Patterns are not easy to recognize when using a command line as it might involve several searches to reveal the desired information.

Table 4.2: Pros and cons of a command line solution

Chapter 5
Discussion

In this section, the solution will be tested and discussed to figure out if a visual representation is a proper way to represent NetFlow data. Testing the proposed solution up against existing command line based tools requires volunteers with knowledge of nfdump. As this is a not a commonly used tool, volunteers from UNINETT were chosen.

With a limited number of test persons, a quantitative examination is not possible. Instead, I performed in-depth qualitative interviews along with testing of both solutions.

As this is not a finished product the testing environment is limited to one month of data, and focused on the potential of the visual solution to detect anomalies and unusual behavior.

5.1 Interviews

During the interviews, the testing environment will be limited. At first the user will be asked of his experience with nfdump. Then the user will try to use the visual representation to find results previously only retrievable through nfdump.

The further focus will be on finding patterns in the data and how this can be done in today's solution versus the proposed solution.

In the last part of the interview there will be a discussion of prepared topics. The template used and some key results can be found in Appendix C

5.2 Discussion topics

5.2.1 Potential in visual solution

Before discussing the solution, it is important to clarify that the proposed solution in this work only covers a small area of use. It does not cover all potential threats and is only an example of how visualization of NetFlow data can be done.

As the discussion proves, nfdump is capable of going more into depth than any visual solution. Although having a significant potential, the visual solution is not able to show very detailed information. But used along with a command line solution, it can prevent the user from performing several high resource demanding nfdump commands, and the visual solution could quickly reveal the information that otherwise would be time-consuming to find with nfdump.

As a part of a larger system, a visual solution has potential. But its lack in diversity requires a lot visual elements to cover all possible attacks.

One of the interview participants had mostly experience using NFsen, a tool that has not been in focus in this work. It creates graphs and gives the option to view flows over different time frames. But when it comes to the specific searches performed, the tool uses nfdump for its possibility for specific searches as used in 2.1.

All in all it is clear that a visual solution brings great contributions to network monitoring, but is facing challenges taking completely over because the level of detail required is narrow, and is often limited to specific events. Both solutions work together and complement each other. As mentioned in 4.2.2, they work well together, and complete functionality where the other is lacking.

Visual representation does not just represent a helpful tool in the security aspect of network monitoring, but presenting statistics of use of the network as well.

5.2.2 Patterns

Discussing the possible use of visualization, its purpose became important, especially what role it would have. In network monitoring patterns are important, especially in detecting anomalies. Changes in behaviour in the network often means a computer or server has been compromised. It was brought up that such network monitoring could be beneficiary in a network in a normal business, but would show to many false positives in a research network as at UNINETT. Spikes in the network that would trigger alarms are not likely to be compromised computers, as the use of the network is not as uniform as other more conform networks in traffic.

5.3 Real data in the proposed solution

During the development of the visual solution only dummy data was used. After development was done, bash scripts found in B was run to extract the necessary data. In 4.1.1 we discovered that the 18th of January between 1300 and 1400 had the highest number of flows. The first section described in 4.2.1, has as its main purpose to point out days with an abnormal amount of flows, and with real data it pointed out 18th of January as well as seen in Figure 5.1. It serves its objective, by showing the bigger picture.

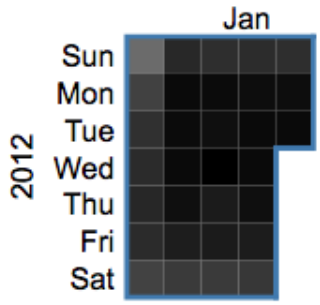

Figure 5.1: Monthly view with real data from UNINETT

Section two of the visual representation, the main element, contains the most data, displaying one hundred different combinations of IP-addresses and ports. Looking at the date pointed out earlier in Figure 5.1, we see that certain combinations is responsible for the largest share of flows in Figure 5.2. Once a specific combination is selected the solution gives the choice to look for patterns by changing the selected date and look for changes in the behaviour on that combination.

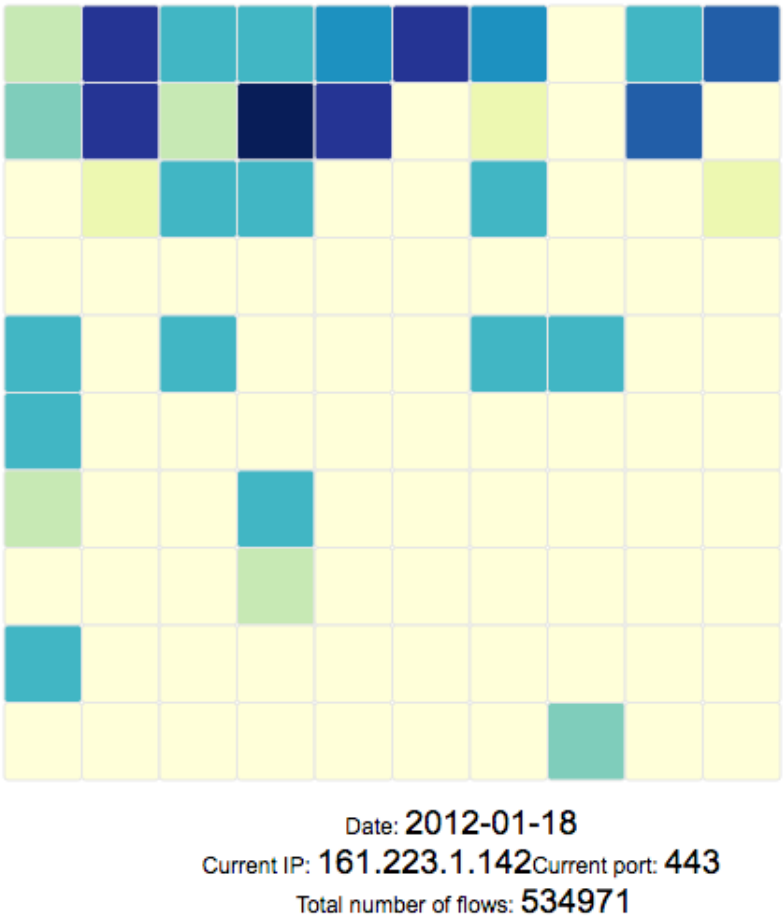

Figure 5.2: Heatmap showing different combinations of IPs and ports

If we consider one of the most significant combinations on the 18th of January, IP address: 161.223.1.142 and Port 443, it is easy to observe the daily patterns compared in Figure 5.3. Showing no abnormal behaviour, we can rule out a DoS attack.

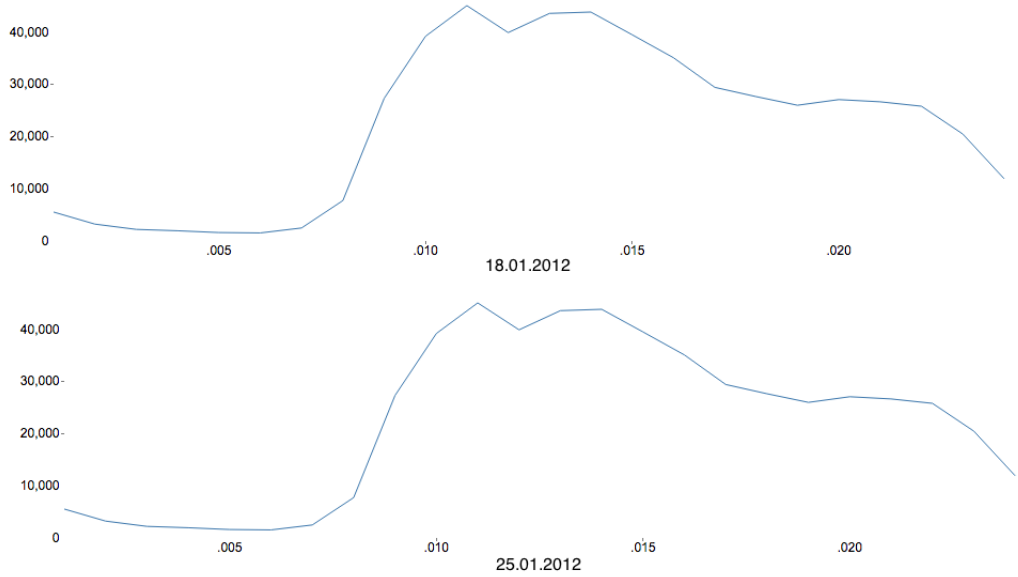

Figure 5.3: Comparison of the pattern between two weekdays on one IP and port combination

When comparing the behaviour in the different combinations of IP-addresses and ports, the last graph is used. It shows how the traffic is distributed over 24-hours, and shows spikes in traffic if there was an attempt of a DoS attack. The traffic can be seen in Figure 5.3 and 5.4.

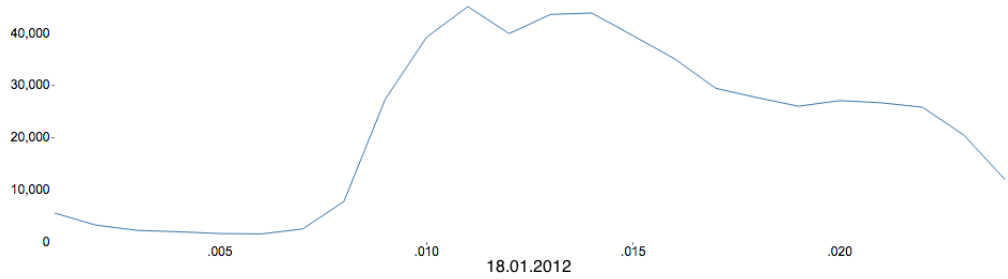

Figure 5.4: Chart showing the traffic on the 18th of January, 2012

Since the dataset did not contain any significant abnormal behaviour, UNINETT provided another month of data where some IP-addresses were responsible for a larger part of the traffic, to see if the solution could reveal something other than reoccurring patterns in the network traffic. The same scripts was executed on the data, and made available to the solution.

This time there is not one certain date that points, which means we need to utilize the second heatmap to spot abnormalities. During a 3-day period between the 17th and 20th of January one specific combination, 79.36.247.140 on port 80, is most used. The traffic starts by growing steadily on the 17th and peaks at 12:00 with over 250,000 flows before it abruptly stops two hours later, shown in Figure 5.5. Using nfdump to find the origin of this traffic it was revealed to come from one single IP-address, 190.118.80.182, and spread across thousands of source ports, shown in Figure 5.6. The reason behind this traffic is unclear as it follows no clear pattern. For example a televised sporting event could give extreme spikes in traffic, but in this case the amount of flows are consistently rising no matter the hour. Without being able to find the applications responsible for the traffic, it is unforunately not possible to conclude with anything other than that this sort of behavior is not normal.

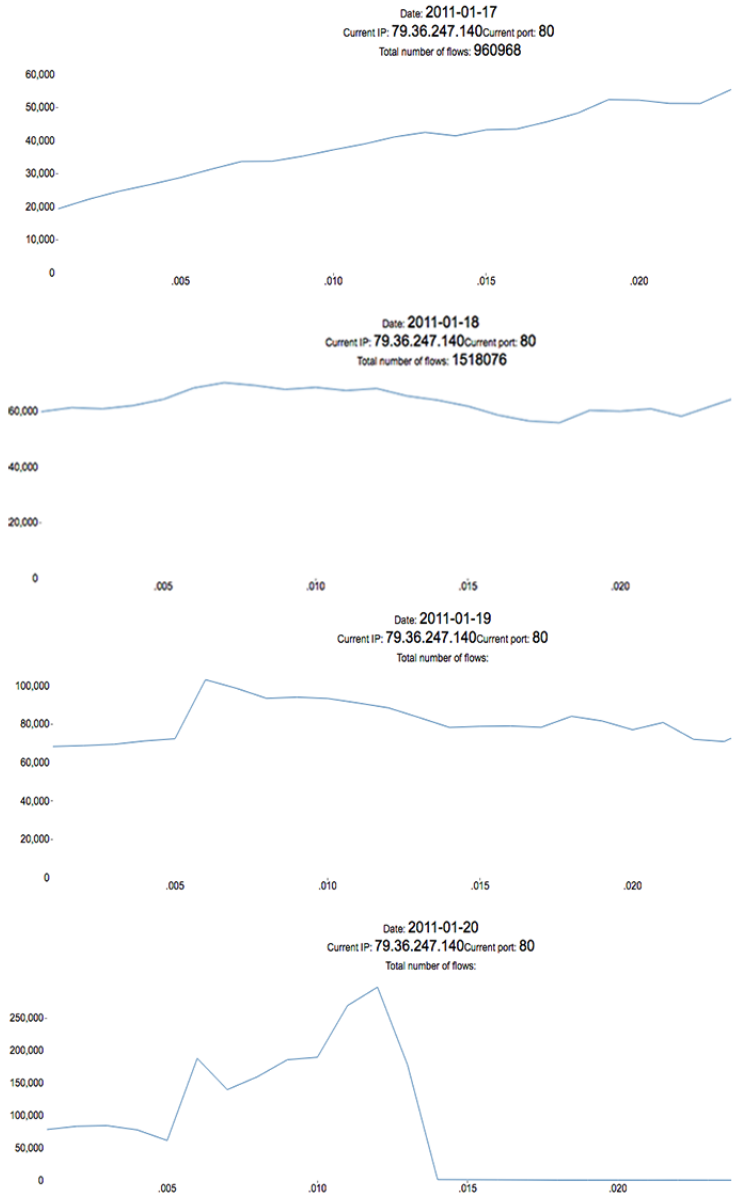

Figure 5.5: Charts displaying the traffic between the 17th and 20th of January for the IP, 79.36.247.140 on port 80

```
eeglarse@iou2:/data/netflow/trd_gw1/2011/01$ nfdump -R /data/netflow/trd_gw1/2011/01/17/nfcapd.201101170000:nfcapd.201101172355 -n 10 -s srcip 'dst ip 79.36.247.140'
Top 10 Src IP Addr ordered by flows:
Date first seen          Duration Proto       Src IP Addr    Flows(%)     Packets(%)       Bytes(%)         pps      bps   bpp
2011-01-16 23:58:38.003 86418.869 any      190.118.80.182   943945(98.2)   949914(98.0)   63.3 M(93.8)       10     5861    66
2011-01-17 00:00:56.455 85432.526 any      192.238.191.74      724( 0.1)     1113( 0.1)   209004( 0.3)        0       19   187
2011-01-17 00:17:50.511 81442.687 any     190.117.229.125      509( 0.1)      564( 0.1)   158936( 0.2)        0       15   281
2011-01-17 07:58:28.855 57634.568 any     161.221.122.162      234( 0.0)      267( 0.0)    51506( 0.1)        0        7   192
2011-01-17 00:19:10.294 78945.810 any      190.117.230.93      213( 0.0)      236( 0.0)   151204( 0.2)        0       15   640
2011-01-17 08:15:32.067 55844.615 any       190.118.206.3      184( 0.0)      201( 0.0)    49325( 0.1)        0        7   245
2011-01-16 23:59:05.191 85420.701 any      190.118.63.173      142( 0.0)      158( 0.0)    22376( 0.0)        0        2   141
2011-01-17 11:29:09.038 44058.819 any       161.221.49.91      139( 0.0)      156( 0.0)    73364( 0.1)        0       13   470
2011-01-17 00:02:37.181 77621.710 any      161.220.23.181      126( 0.0)      134( 0.0)    57520( 0.1)        0        5   429
2011-01-17 00:15:39.858 78647.718 any       161.221.8.233      120( 0.0)      141( 0.0)    38133( 0.1)        0        3   270

Summary: total flows: 960972, total bytes: 67.5 M, total packets: 969195, avg bps: 6247, avg pps: 11, avg bpp: 69
Time window: <unknown>
Total flows processed: 112055254, Blocks skipped: 0, Bytes read: 5826958276
Sys: 14.626s flows/second: 7661119.3  Wall: 24.067s flows/second: 4655963.5
eeglarse@iou2:/data/netflow/trd_gw1/2011/01$ nfdump -R /data/netflow/trd_gw1/2011/01/17/nfcapd.201101170000:nfcapd.201101172355 -n 10 -s srcport 'dst ip 79.36.247.14
Top 10 Src Port ordered by flows:
Date first seen          Duration Proto          Src Port    Flows(%)     Packets(%)       Bytes(%)         pps      bps   bpp
2011-01-17 01:12:10.021 81538.869 any             60880       63( 0.0)       63( 0.0)     4230( 0.0)        0        0    67
2011-01-17 00:55:05.918 81118.955 any             49834       59( 0.0)       60( 0.0)     4690( 0.0)        0        0    78
2011-01-17 00:38:05.331 83889.055 any             41255       58( 0.0)       58( 0.0)     3877( 0.0)        0        0    66
2011-01-17 01:39:53.009 78877.868 any             56328       58( 0.0)       59( 0.0)     5528( 0.0)        0        0    93
2011-01-17 00:51:28.810 82676.460 any             57277       58( 0.0)       58( 0.0)     3723( 0.0)        0        0    64
2011-01-17 00:16:09.098 85126.951 any             47422       57( 0.0)       57( 0.0)     3955( 0.0)        0        0    69
2011-01-17 00:04:28.557 85584.835 any             55919       56( 0.0)       57( 0.0)     5960( 0.0)        0        0   104
2011-01-17 00:38:45.677 82802.428 any             44546       56( 0.0)       56( 0.0)     3577( 0.0)        0        0    63
2011-01-17 00:33:35.253 84112.783 any             55569       55( 0.0)       55( 0.0)     3519( 0.0)        0        0    63
2011-01-17 00:39:37.590 83152.213 any             57486       55( 0.0)       57( 0.0)     3303( 0.0)        0        0    57

Summary: total flows: 960972, total bytes: 67.5 M, total packets: 969195, avg bps: 6247, avg pps: 11, avg bpp: 69
Time window: <unknown>
Total flows processed: 112055254, Blocks skipped: 0, Bytes read: 5826958276
Sys: 7.199s flows/second: 15563435.1 Wall: 7.224s flows/second: 15511032.9
```

Figure 5.6: nfdump screenshot showing source ports and IPs to 79.36.247.140 on the 17th of January

Other irregularities appear on the days between the 13th through 17th of January, a series of days where 9 out of 10 ports receive traffic on one specific IP-address.The first day is shown in Figure 5.7 And the single port with no traffic is port 0, which is reserved [24]. All of this traffic originated from one singe IP-address and port as seen in Figure 5.8.

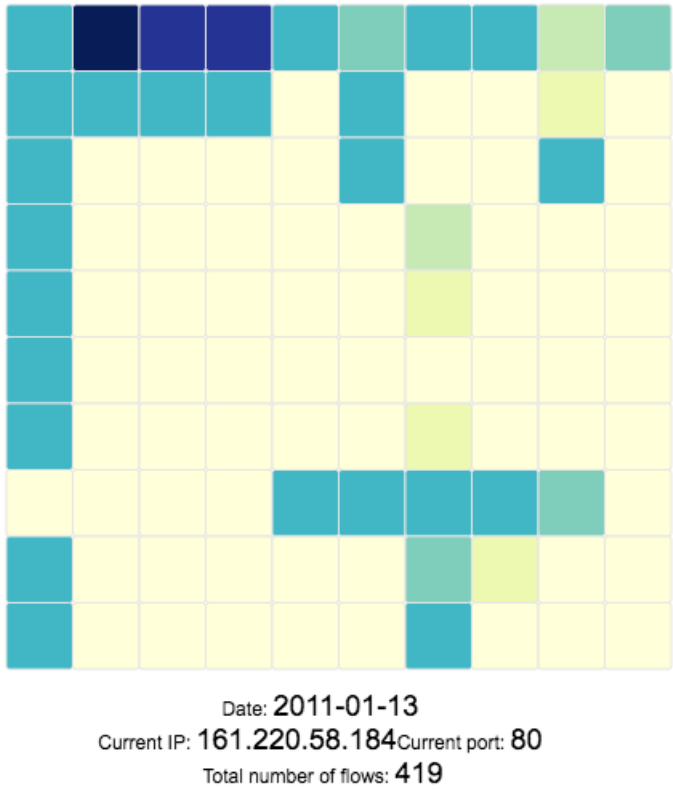

Figure 5.7: Heatmap displaying traffic across multiple ports on a single IP-address

```
eeglarse@iou2:/data/netflow/trd_gw1/2011/01$ nfdump -R /data/netflow/trd_gw1/2011/01/15/nfcapd.201101150000:nfcapd.201101152355 -n 10 -s srcip 'dst ip 161.220.58.184'
Top 10 Src IP Addr ordered by flows:
Date first seen          Duration Proto       Src IP Addr    Flows(%)     Packets(%)       Bytes(%)         pps      bps   bpp
2011-01-14 23:58:57.022 86420.829 any      138.170.71.211   35.9 M(100.0)   45.3 M(100.0)   25.9 G(100.0)      524    2.4 M   570
2011-01-15 16:02:42.123 28352.892 any      212.164.119.246      291( 0.0)      301( 0.0)    30803( 0.0)        0        8   102
2011-01-15 10:00:17.561  4902.501 any        107.61.98.96       180( 0.0)      478( 0.0)    76480( 0.0)        0      124   160
2011-01-15 09:56:26.722  4958.397 any       69.100.163.125      148( 0.0)      638( 0.0)   320992( 0.0)        0      517   503
2011-01-15 04:18:29.390 70806.121 any        94.201.72.208       13( 0.0)       20( 0.0)     1350( 0.0)        0        0    67
2011-01-15 06:24:01.371 53999.428 any       94.215.105.255        6( 0.0)        6( 0.0)      325( 0.0)        0        0    54
2011-01-15 14:52:09.570    64.255 any       134.29.73.115         5( 0.0)        7( 0.0)      352( 0.0)        0       43    50
2011-01-15 03:44:15.954 56540.892 any       95.165.112.101        4( 0.0)        4( 0.0)      208( 0.0)        0        0    52
2011-01-15 22:13:23.794     0.185 any         94.201.83.63        3( 0.0)        3( 0.0)      496( 0.0)       16    21448   165
2011-01-15 19:12:15.282 15349.276 any      190.119.194.162        2( 0.0)        2( 0.0)      152( 0.0)        0        0    76

Summary: total flows: 35893747, total bytes: 25.9 G, total packets: 45.3 M, avg bps: 2.4 M, avg pps: 524, avg bpp: 570
Time window: <unknown>
Total flows processed: 114088220, Blocks skipped: 0, Bytes read: 5932674068
Sys: 16.920s flows/second: 6742769.5  Wall: 25.436s flows/second: 4485154.3
eeglarse@iou2:/data/netflow/trd_gw1/2011/01$ nfdump -R /data/netflow/trd_gw1/2011/01/15/nfcapd.201101150000:nfcapd.201101152355 -n 10 -s srcport 'dst ip 161.220.58.184'
Top 10 Src Port ordered by flows:
Date first seen          Duration Proto          Src Port    Flows(%)     Packets(%)       Bytes(%)         pps      bps   bpp
2011-01-14 23:58:57.022 86420.829 any              48761   35.9 M(100.0)   45.3 M(100.0)   25.9 G(100.0)      524    2.4 M   570
2011-01-15 09:56:26.722  4958.397 any              43954      148( 0.0)      638( 0.0)   320992( 0.0)        0      517   503
2011-01-15 10:00:17.561  4902.501 any              60090      134( 0.0)      354( 0.0)    51387( 0.0)        0       83   145
2011-01-15 10:38:18.849  1524.928 any              33404       46( 0.0)      124( 0.0)    25093( 0.0)        0      131   202
2011-01-15 06:24:01.371 53999.428 any                 80        7( 0.0)        7( 0.0)     1825( 0.0)        0        0   260
2011-01-15 16:08:22.430     0.426 any              44582        2( 0.0)        2( 0.0)      133( 0.0)        4     2497    66
2011-01-15 18:52:09.217     5.404 any              38734        2( 0.0)        2( 0.0)      208( 0.0)        0      307   104
2011-01-15 03:44:15.954 47920.062 any              57355        2( 0.0)        2( 0.0)      204( 0.0)        0        0   102
2011-01-15 18:50:12.582     0.656 any              58269        2( 0.0)        2( 0.0)      141( 0.0)        3     1719    70
2011-01-15 22:10:32.857    10.069 any              45878        2( 0.0)        2( 0.0)      156( 0.0)        0      123    78

Summary: total flows: 35893747, total bytes: 25.9 G, total packets: 45.3 M, avg bps: 2.4 M, avg pps: 524, avg bpp: 570
Time window: <unknown>
Total flows processed: 114088220, Blocks skipped: 0, Bytes read: 5932674068
Sys: 8.293s flows/second: 13756559.0 Wall: 8.319s flows/second: 13713075.3
```

Figure 5.8: nfdump screenshot showing source ports and IPs to 161.220.58.184 on the 15th of January

Chapter 6
Challenges

6.1 Large data sets

When visualizing big data the main challenge is to effectively show the core message of the data. Considering one hour of the data provided from UNINETT, there are almost 400,000 different IP-adresses, and the amounts of flows is in the millions.

In section 2.2.1 good visualization is said to be able to present many numbers in a small space, make large data sets coherent, and reveal data at several levels of detail. I chose to create individual modules with D3.js, with each covering a different layer of detail.

6.1.1 IP-spectrum

As mentioned the range of the Internet Protocol version 4 (IPv4) is large, and with the emergence of IPv6 there is the challenge to represent such a large spectrum as seen in6.1.1. In 4.2.1 this was resolved with pre-processing of the data for a specific task. In other cases such a limitation on the number of IP-addresses represented wouldn't be satisfying. A comparison of the size of the two IP versions is done in table 6.1.1

6.1.2 Increasing number of flows

The amount of data sent these days are expanding quickly. This means the number of flows will follow, and a visual solution will need to be scalable to handle this increase. In the solution in 4.2 the first overview could scale as the heatmap takes in consideration the range of total flows. The second part of the solution is capable of handling larger amounts of flows, but the limitation in the number of ports and IP-addresses could limit certain attacks as port scanning. With the increase in IPv6, the visual representation itself could become less intuitive and simple as the possible combinations of IP-addresses and ports would be too big to yield results(, or reveal

IPv4	IPv6
Address size: 32-bit number	**Address size:** 128-bit number
Address format: Decimal notation: 192.168.0.0	**Address format:** Hexadecimal notation: 3FFF:F200:0234:AB00:0123:4567:8901:ABCD
Number of addresses: $2^{32} = 4,294,967,296$	**Number of addresses:** $2^{128} = 340,282,366,920,938,463,463,374,607,431,768,211,456$

Table 6.1: IPv4 vs. IPv6

attacks). As the last graph is based on simply number of flows over time, it could prove effective even as the number of flows increase. It could be harder to reveal minor spikes in flows as it will be less distinguished than before.

6.2 Live updates

During the development of the solution made in this work the data has been static and the time used to process the data and present it has not been taken into consideration. If a visual solution is to be used to monitor traffic as it is being collected, the processing of the live data needs to be improved to be able to detect attacks and deviations as they are happening. nfdump is a powerful tool, but the amounts of data generated are so large, and will increase further, that the possibility to filter out specific information is to time consuming and require a lot of resources from UNINETT. It was not made a priority in this work as the functionality of the visual solution up against nfdump was the main focus, not if it met requirements in speed, live monitoring and searching within the data.

6.3 Areas of use

In this work it is recognized that there are several positive aspects of a visual solution. But to be as effective as it can be, it needs to work well along with other tools. In 2.1.3 the current tool, nfsen, is used in combination with other solutions. The same combination could be just as effective used along with a more updated and interactive visual solution as developed in this work.

Alone it can not serve as a complete solution, but as a stand alone product it can show important statistics about the network which can be used in anything from

security measures to allocation of bandwidth. In [25] they set up an SQL-database storing a large numbers of NetFlows to be able to perform queries to provide usage statistics and network forensics, as well as the intrusion detection. Showing that the purpose of data mining NetFlow packets can prove very useful within many areas of use.

Chapter 7
Concluding Remarks and Future Work

In this work the theory behind both Cisco's NetFlow and basic visualization theory has been presented. This has been used as background for the novel part of the work which has been the development and testing of a simple tool to interact with the NetFlow data to find unusual behavior in the network, revealing several abnormalities described in 5.3. Potential in such a solution has been discussed through qualitative interviews with experienced users of the current tools. Results show there is a need for developments within the field of visualization in network monitoring.

What has been created in this work is merely a fraction of what is possible with both NetFlow data and D3.js. Further analysis of what information that should be included in the visual presentation could give a solution capable of discovering anomalies not being picked up by current systems. By showing how visual elements can improve how people interact with NetFlow data I hope the potential for using visual tools combined with human inspection and experience has been accentuated.

7.1 Recommendations for Future Work

In the work done in this paper only the basic knowledge of how NetFlow can be visualized has been done. To further improve its functionality and operation in-depth research into performance is required.

7.1.1 Further testing

The data provided from UNINETT was not confirmed to contain any attacks, but showed although that the visual representation of the data revealed similar patterns as discovered trough the current nfdump solution. Future work should test the visual solution against larger sets of data to prove scalability and the ability to reveal patterns over bigger time slots, and larger sets of data.

7.1.2 Customized searches

As showed in section 4.1.1 the commands in nfdump can reveal detailed information in few commands, but there are still a few searches required to obtain the requested data. By creating an Application Programming Interface (API) that simplifies the extraction of the information used in the visual elements, the possibility to display live NetFlow data is obtainable.

7.1.3 Expand visual elements

Pre-processing data from UNINETT was time consuming and the focus was to display an example of what is possible. With a faster way of searching and processing the data as mentioned in the previous section it could have been expanded to include a wider scope. The possibility to choose between source and destination IP-addresses and ports would be more effective in monitoring the traffic, since it would make the user able to see who is originating as well as who is receiving the traffic. The amount of IP-addresses could be higher to paint a clearer picture. An example of a solution with data from a year, and 24 IP-addresses, and a weekly graph along with the current 24-hour graph is shown in Figure 7.1.

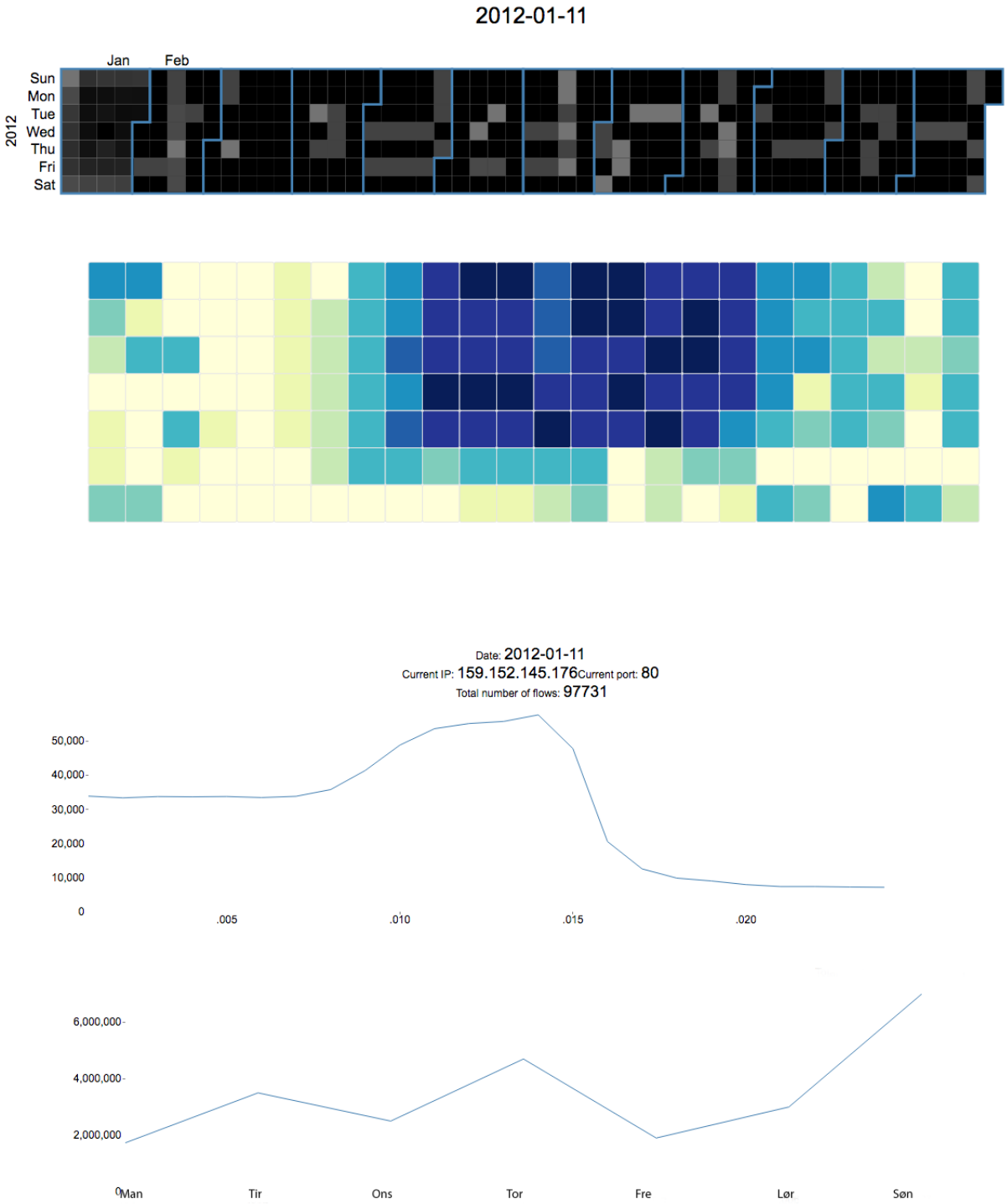

Figure 7.1: Example of the potential of a visual solution

References

[1] P. Services, C. Software, C. Technologies, M. Instrumentation, C. NetFlow, D. Literature, and W. Papers, "Introduction to cisco ios netflow - a technical overview," 2016.

[2] K. Lakkaraju, W. Yurcik, R. Bearavolu, and A. J. Lee, "Nvisionip: an interactive network flow visualization tool for security," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 3, pp. 2675–2680 vol.3, Oct 2004.

[3] T. Prolexic, "Distributed re ection denial of service (drdos) attacks an introduction to the drdos white paper series," 2013.

[4] "A cisco guide to defending against distributed denial of service attacks," 2016.

[5] M. Bostock, "D3.js - data-driven documents," 2016.

[6] d. (Username), "Update d3.js data with button press," 2015.

[7] K. G. Coffman and A. M. Odlyzko, *Handbook of Massive Data Sets*, ch. Internet Growth: Is There a "Moore's Law" for Data Traffic?, pp. 47–93. Boston, MA: Springer US, 2002.

[8] G. H. Jamieson, *Visual communication: More than meets the eye*. Intellect Books, 2007.

[9] M. Polychronakis, E. P. Markatos, K. G. Anagnostakis, and A. Øslebø, "Design of an application programming interface for ip network monitoring," in *Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP*, vol. 1, pp. 483–496, IEEE, 2004.

[10] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better netflow," *SIGCOMM Comput. Commun. Rev.*, vol. 34, pp. 245–256, Aug. 2004.

[11] "Nfdump.sourceforge.net," 2016.

[12] B. Claise, "Cisco systems netflow services export version 9," 2004.

[13] B. Trammell and E. Boschi, "An introduction to ip flow information export (ipfix)," *Communications Magazine, IEEE*, vol. 49, no. 4, pp. 89–95, 2011.

[14] *Computers & Mathematics with Applications*, vol. 37, no. 10, p. 173, 1999.

[15] "Rfc 3917," 2016.

[16] V. Friedman, "Data visualization and infographics smashing magazine," 2008.

[17] A. McAfee, E. Brynjolfsson, T. H. Davenport, D. Patil, and D. Barton, "Big data," *The management revolution. Harvard Bus Rev*, vol. 90, no. 10, pp. 61–67, 2012.

[18] E. R. Tufte and P. Graves-Morris, *The visual display of quantitative information*, vol. 2. Graphics press Cheshire, CT, 1983.

[19] "Data presentation architecture," 2016.

[20] "Simple d3.js graph," 2016.

[21] K. Lakkaraju, W. Yurcik, and A. J. Lee, "Nvisionip: Netflow visualizations of system state for security situational awareness," in *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, VizSEC/DMSEC '04, (New York, NY, USA), pp. 65–72, ACM, 2004.

[22] J. Zhihua, "Visualization of network traffic to detect malicious network activity," 2008.

[23] N. Downie, "Chart. js documentation," *Dostopno na: http://www. chartjs. org/-docs (marec 2014)*, vol. 65, p. 66, 2014.

[24] B. Mitchell, "Tcp/udp port 0," 2016.

[25] J.-P. Navarro, B. Nickless, and L. Winkler, "Combining cisco netflow exports with relational database technology for usage statistics, intrusion detection, and network forensics," in *Proceedings of the 14th Large Installation Systems Administration Conference (LISA 2000)*, pp. 285–290, 2000.

Appendix

Appendix A

```
<!DOCTYPE html>
<meta charset="utf-8">
<style> /* set the CSS */

body { font: 24px Arial;}

path {
    stroke: steelblue;
    stroke-width: 2;
    fill: none;
}

.axis path,
.axis line {
    fill: none;
    stroke: grey;
    stroke-width: 1;
    shape-rendering: crispEdges;
}


rect.bordered {
        stroke: #E6E6E6;
        stroke-width:2px;
      }

      text.mono {
        font-size: 12pt;
        font-family: Consolas, courier;
        fill: #aaa;
      }

      text.axis-workweek {
        fill: #000;
      }

      text.axis-worktime {
        fill: #000;
```

```
39          }
            .RdYlGn .q0-11{fill:rgb(165,0,38)}
41  .RdYlGn .q1-11{fill:rgb(215,48,39)}
    .RdYlGn .q2-11{fill:rgb(244,109,67)}
43  .RdYlGn .q3-11{fill:rgb(253,174,97)}
    .RdYlGn .q4-11{fill:rgb(254,224,139)}
45  .RdYlGn .q5-11{fill:rgb(255,255,191)}
    .RdYlGn .q6-11{fill:rgb(217,239,139)}
47  .RdYlGn .q7-11{fill:rgb(166,217,106)}
    .RdYlGn .q8-11{fill:rgb(102,189,99)}
49  .RdYlGn .q9-11{fill:rgb(26,152,80)}
    .RdYlGn .q10-11{fill:rgb(0,104,55)}
51
    .header {
53      height:50px;
        background:#F0F0F0;
55      border:1px solid #CCC;
        width:960px;
57      margin:0px auto;
    }
59  </style>
    <body>
61
    <!-- load the d3.js library -->
63  <script src="http://d3js.org/d3.v3.min.js"></script>
    <script type='text/javascript' src='knockout-min.js'></script>
65  <div>
    <div style="font-size: 50px; text-align: center;margin-top: 20px;margin
        -bottom: 20px"data-bind="text: currentDate()"></div>
67  <div id="year" style="margin-top=20px; font-size: 12px; "></div>
    <div id="area1" style="padding-left: -50px"></div>
69
    <div style="text-align: center">
71  Date: <span style="text-align: center; font-size: 36px;"data-bind="text
        : currentDate()"></span>

73  <br>
    Current IP: <span style="text-align: center; font-size: 36px;"data-bind
        ="text: chosenIp()"></span>Current port: <span style="text-align:
        center;font-size: 36px;"data-bind="text: chosenPort()"></span>
75  <br>
    Total number of flows: <span style="text-align: center; font-size: 36
        px;"data-bind="text: totalFlows()">
77  </div>
    <div id="area2"></div>
79  </div>

81  <script>

83  function AppViewModel(){
      this.currentDay = ko.observable(1);
85    this.availableCountries = ko.observableArray([]);
```

```
  this.availablePorts = ko.observableArray();
  this.chosenIp = ko.observable('159.152.145.176');
  this.chosenPort = ko.observable('80');
  this.currentYear = ko.observable(2012);
  this.currentMonth = ko.observable(1);
  this.currentDate = ko.observable('2012-01-11');
  this.totalFlows = ko.observable('97731');
  this.flows = ko.observable();
}

var parseDate = d3.time.format("%d-%b-%y").parse;

ko.applyBindings(AppViewModel);


function readTextFile(file)
{
    var rawFile = new XMLHttpRequest();
    rawFile.open("GET", file, true);
    rawFile.onreadystatechange = function ()
    {
        if(rawFile.readyState === 4)
        {
            if(rawFile.status === 200 || rawFile.status == 0)
            {
                var allText = rawFile.responseText;
                readIps(allText);            }
        }
    }
    rawFile.send(null);
}

function readIps(text){
  list = text.split(",");
  availableCountries(list);
  console.log(availableCountries())
}

readTextFile("2012_01_01.txt")

function firstGraph(){
  var width = 900,
    height = 100,
    cellSize = 15; // cell size
    week_days = ['Sun','Mon','Tue','Wed','Thu','Fri','Sat']
    month = ['Jan','Feb']

var day = d3.time.format("%w"),
    week = d3.time.format("%U"),
    percent = d3.format(".1%"),
  format = d3.time.format("%Y-%m-%d");
  parseDate = d3.time.format("%Y-%m-%d").parse;
```

```
var color = d3.scale.linear().range(["white", 'black'])
    .domain([0, 10])

var svg = d3.select("#year").selectAll("svg")
    .data(d3.range(2012, 2013))
  .enter().append("svg")
    .attr("width", '100%')
    .attr("data-height", '0.5678')
    .attr("viewBox",'0 0 900 105')
    .attr("class", "RdYlGn")
    .style("height","450")

  .append("g")
    .attr("transform", "translate(" + ((width - cellSize * 53) / 2) + "
    ," + (height - cellSize * 7 - 1) + ")");

svg.append("text")
    .attr("transform", "translate(-38," + cellSize * 3.5 + ")rotate
    (-90)")
    .style("text-anchor", "middle")
    .text(function(d) { return d; });

for (var i=0; i<7; i++)
{
svg.append("text")
    .attr("transform", "translate(-5," + cellSize*(i+1) + ")")
    .style("text-anchor", "end")
    .attr("dy", "-.25em")
    .text(function(d) { return week_days[i]; });
 }

var rect = svg.selectAll(".day")
    .data(function(d) { return d3.time.days(new Date(d, 0, 1), new Date
    (d + 1, 0, 1)); })
  .enter()
  .append("rect")
    .attr("class", "day")
    .attr("width", cellSize)
    .attr("height", cellSize)
    .attr("x", function(d) { return week(d) * cellSize; })
    .attr("y", function(d) { return day(d) * cellSize; })
    .attr("fill",'#fff')
    .attr("title", function(d) { return "value : "+d.Comparison_Type})
    .datum(format)
    .on("click", function(d){
      heatmapChart("02/heatmap_"+d+".csv");
      currentDate(d);
      updateGraph();
      totalFlows('');
    });
```

```
var legend = svg.selectAll(".legend")
      .data(month)
    .enter().append("g")
      .attr("class", "legend")
      .attr("transform", function(d, i) { return "translate(" + (((i+1)
     * 50)+8) + ",0)"; });

legend.append("text")
   .attr("class", function(d,i){ return month[i] })
   .style("text-anchor", "end")
   .attr("dy", "-.25em")
   .text(function(d,i){ return month[i] });

svg.selectAll(".month")
    .data(function(d) { return d3.time.months(new Date(d, 0, 1), new
    Date(d+1 , 0, 1)); })
  .enter().append("path")
    .attr("class", "month")
    .attr("id", function(d,i){ return month[i] })
    .attr("d", monthPath);

d3.csv("datefile.csv", function(error, csv) {

  csv.forEach(function(d) {
    d.Comparison_Type = parseInt(d.Comparison_Type);
  });

 var Comparison_Type_Max = d3.max(csv, function(d) { return d.
    Comparison_Type; });

  var data = d3.nest()
    .key(function(d) { return d.Date; })
    .rollup(function(d) { return  10*(Math.sqrt(d[0].Comparison_Type /
    Comparison_Type_Max)); })
    .map(csv);

  rect.filter(function(d) { return d in data; })
      .attr("fill", function(d) { return color(data[d]); })
    .attr("title", function(d) { return "value : "+data[d].
    Comparison_Type});

});

function numberWithCommas(x) {
    x = x.toString();
    var pattern = /(-?\d+)(\d{3})/;
    while (pattern.test(x))
        x = x.replace(pattern, "$1,$2");
    return x;
}

function monthPath(t0) {
```

```
  var t1 = new Date(t0.getFullYear(), t0.getMonth() + 1, 0),
      d0 = +day(t0), w0 = +week(t0),
      d1 = +day(t1), w1 = +week(t1);
  return "M" + (w0 + 1) * cellSize + "," + d0 * cellSize
      + "H" + w0 * cellSize + "V" + 7 * cellSize
      + "H" + w1 * cellSize + "V" + (d1 + 1) * cellSize
      + "H" + (w1 + 1) * cellSize + "V" + 0
      + "H" + (w0 + 1) * cellSize + "Z";
}

d3.select(self.frameElement).style("height", "2910px");

var margin = { top: 50, right: 0, bottom: 500, left: 200 },
          width = 2200 - margin.left - margin.right,
          height = 900 - margin.top - margin.bottom,
          gridSize = Math.floor(width / 24),
          legendElementWidth = gridSize,
          buckets = 9,
          colors = ["#ffffd9","#edf8b1","#c7e9b4","#7fcdbb","#41b6c4","#1d91c0","#225ea8","#253494","#081d58"], // alternatively colorbrewer.YlGnBu[9]
          days = availablePorts(),
          times = availableCountries(),
          datasets = ["heatmap/data.tsv", "heatmap/data2.tsv"];

      var svg = d3.select("#area1")
          .append("svg")
          .attr("width", width + margin.left + margin.right)
          .attr("height", height + margin.top + margin.bottom)
          .append("g")
          .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

      var dayLabels = svg.selectAll(".dayLabel")
          .data(availableCountries())
          .enter().append("text")
            .text(function (d) { return d; })
            .attr("x", 0)
            .attr("y", function (d, i) { return i * gridSize; })
            .style("text-anchor", "end")
            .attr("transform", "translate(-6," + gridSize / 1.5 + ")")
            .attr("class", function (d, i) { return ((i >= 0 && i <= 4) ? "dayLabel mono axis axis-workweek" : "dayLabel mono axis"); })
            .transition().duration(250);


      var timeLabels = svg.selectAll(".timeLabel")
          .data(availableCountries())
          .enter().append("text")
            .text(function(d) { return d; })
            .attr("x", function(d, i) { return i * gridSize; })
            .attr("y", 0)
```

```
            .style("text-anchor", "middle")
            .attr("transform", "translate(" + gridSize / 2 + ", -6)")
            .attr("class", function(d, i) { return ((i >= 7 && i <= 16)
 ? "timeLabel mono axis axis-worktime" : "timeLabel mono axis"); })
;

  var heatmapChart = function(csvFile) {
    d3.tsv(csvFile,
    function(d) {
      return {
        day: +d.day,
        hour: +d.hour,
        value: +d.value,
        ip: d.ip,
        port: d.port
      };
    },
    function(error, data) {
      var colorScale = d3.scale.quantile()
          .domain([0, buckets - 1, d3.max(data, function (d) {
return d.value; })])
          .range(colors);

      var cards = svg.selectAll(".hour")
          .data(data, function(d) {return d.day+':'+d.hour;});

      cards.append("title");

      cards.enter().append("rect")
          .attr("x", function(d) { return (d.hour - 1) * gridSize;
})
          .attr("y", function(d) { return (d.day - 1) * gridSize;
})
          .attr("rx", 4)
          .attr("ry", 4)
          .attr("class", "hour bordered")
          .attr("width", gridSize)
          .attr("height", gridSize)
          .style("fill", colors[0])
          .on("click", function(d){
            chosenIp(d.ip);
            chosenPort(d.port);
            totalFlows(d.value);
            console.log(d.value+" "+""+chosenIp()+" "+chosenPort())
;
            updateGraph();
          });

      cards.transition().duration(1500)
          .style("fill", function(d) { return colorScale(d.value);
});
```

```
327             cards.select("title").text(function(d) { return d.ip +' and '
    + d.port+': '+d.value; });

329             cards.exit().remove();

331           });
        };
333
        heatmapChart('heatmap_temp.tsv');
335
        var datasetpicker = d3.select("#dataset-picker").selectAll(".
    dataset-button")
337         .data(datasets);

339       datasetpicker.enter()
          .append("input")
341       .attr("value", function(d){ return "Dataset " + d })
          .attr("type", "button")
343       .attr("class", "dataset-button")
          .on("click", function(d) {
345         updateGraph();
          });
347
function updateGraph(){
349 d3.select("#area2").selectAll("svg").remove();

351 var margin = {top: 30, right: 20, bottom: 30, left: 200},
    width = 2000 - margin.left - margin.right,
353 height = 500 - margin.top - margin.bottom;

355 // Parse the date / time


357
    // Set the ranges
359 var x = d3.time.scale().range([0, width]);
    var y = d3.scale.linear().range([height, 0]);
361
    // Define the axes
363 var xAxis = d3.svg.axis().scale(x)
        .orient("bottom").ticks(5);
365
    var yAxis = d3.svg.axis().scale(y)
367     .orient("left").ticks(5);

369 // Define the line
    var valueline = d3.svg.line()
371     .x(function(d) { return x(d.date); })
        .y(function(d) { return y(d.close); });
373
    // Adds the svg canvas
375 var svg = d3.select("#area2")
        .append("svg")
```

```
377         .attr("width", width + margin.left + margin.right)
            .attr("height", height + margin.top + margin.bottom)
379     .append("g")
            .attr("transform",
381               "translate(" + margin.left + "," + margin.top + ")");

383 // Get the data
d3.csv("/chart/chart/"+currentDate()+'/'+currentDate()+"_"+chosenIp()+'
    _'+chosenPort()+'.csv', function(error, data) {
385     data.forEach(function(d) {
            d.date = +d.date;
387         d.close = +d.close;
        });
389
        // Scale the range of the data
391     x.domain(d3.extent(data, function(d) { return d.date; }));
        y.domain([0, d3.max(data, function(d) { return d.close; })]);
393
        // Add the valueline path.
395     svg.append("path")
            .attr("class", "line")
397         .attr("d", valueline(data));

399     // Add the X Axis
        svg.append("g")
401         .attr("class", "x axis")
            .attr("transform", "translate(0," + height + ")")
403         .call(xAxis);

405     // Add the Y Axis
        svg.append("g")
407         .attr("class", "y axis")
            .call(yAxis);
409
      });
411 };

413 updateGraph()
  };
415

function changeIpOrPort() {
417
    // Get the data again
419 d3.csv("2011/"+currentDate()+'/'+currentDate()+"_"+chosenIp()+'_'+
    chosenPort()+'.csv', function(error, data) {
        data.forEach(function(d) {
421         d.date = parseDate(d.date);
            d.close = +d.close;
423       });

425       // Scale the range of the data again
          x.domain(d3.extent(data, function(d) { return d.date; }));
```

```
427         y.domain([0, d3.max(data, function(d) { return d.close; })]);

429     // Select the section we want to apply our changes to
        var svg = d3.select("#area2").transition();
431
        // Make the changes
433         svg.select(".line")   // change the line
                .duration(750)
435             .attr("d", valueline(data));
            svg.select(".x.axis") // change the x axis
437             .duration(750)
                .call(xAxis);
439         svg.select(".y.axis") // change the y axis
                .duration(750)
441             .call(yAxis);

443     });
    }
445
    firstGraph();
447 </script>
    </body>
```

Listing A.1: HTML and scripts to create the proposed solution

Appendix

Appendix B

Appendix B contains the scripts used to create .csv files from all the data made available from UNINETT.

A script that creates .csv files for every nfcapd file in a day. This script is run by another short script that runs it 31 times for each day.

```
\label{csv_daily}
#!/bin/bash
mkdir /home/eeglarse/flowtest/2012_02/$(printf "%02d" $1)
clock_converter(){
  if [ $(($1 % 100)) -gt 59 ]; then
    return $(($1 % 100))
  fi
  if [ $(($1 % 100)) -lt 60 ]; then
    echo $(printf "%04d" $1)
    return $(($1 % 100))
  fi
  }

for (( c=0; c<=2355; c += 5 ))
do
   nfdump -r $(printf "%02d" $1)/nfcapd.201202$(printf "%02d" $1)$(
    clock_converter $c ) -n 10 -s srcip -o csv > /home/eeglarse/
    flowtest/2012_02/$(printf "%02d" $1)/$( clock_converter $c ).csv
done
```

Listing B.1: Creates .csv files for every nfcapd file in a day

A script that fetches the total amount of flows for each day and creates a file with the values.

```


clock_converter(){
  if [ $(($1 % 100)) -gt 59 ]; then
    return $(($1 % 100))
```

```
  fi
  if [ $(($1 % 100)) -lt 60 ]; then
    echo $(printf "%04d" $1)
    return $(($1 % 100))
  fi
  }


for (( c=0; c<=2355; c += 5 ))
do
   total_file=$(awk -F',' 'NR == 15 { print $1}' /home/eeglarse/
    flowtest/2012_02/$(printf "%02d" $1)/$( clock_converter $c ).csv)
   echo $total_file >> testfile2$1.csv
done

awk '{s+=$1} END {print s}' testfile2$1.csv >>datefile2.csv
```

Listing B.2: Total amount of flows for each day

A script that finds the top 10 used IP-adresses for each day.

```

nfdump -R /data/netflow/oslo_gw/2012/01/$(printf "%02d" $1)/nfcapd
    .201201$(printf "%02d" $1)0000:nfcapd.201201$(printf "%02d" $1)2355
    -n 10 -s dstip -o csv > /home/eeglarse/flowtest/top10/$(printf "
    %02d" $1).csv
```

Listing B.3: Top 10 used IP-adresses for each day

A script that creates a list of the top 10 most popular ports, based on the 10 most popular IP-addresses.

```
ip_string=''
ip=$(awk -F',' 'NR == 2 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $1).csv)
ip_string+='dst ip '$ip' or '
ip=$(awk -F',' 'NR == 3 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $1).csv)
ip_string+='dst ip '$ip' or '
ip=$(awk -F',' 'NR == 4 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $1).csv)
ip_string+='dst ip '$ip' or '
ip=$(awk -F',' 'NR == 5 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $1).csv)
ip_string+='dst ip '$ip' or '
ip=$(awk -F',' 'NR == 6 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $1).csv)
ip_string+='dst ip '$ip' or '
ip=$(awk -F',' 'NR == 7 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $1).csv)
ip_string+='dst ip '$ip' or '
```

```
ip=$(awk -F',' 'NR == 8 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $1).csv)
ip_string+='dst ip '$ip' or '
ip=$(awk -F',' 'NR == 9 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $1).csv)
ip_string+='dst ip '$ip' or '
ip=$(awk -F',' 'NR == 10 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $1).csv)
ip_string+='dst ip '$ip' or '
ip=$(awk -F',' 'NR == 11 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $1).csv)
ip_string+='dst ip '$ip


nfdump -R /data/netflow/oslo_gw/2012/01/$(printf "%02d" $1)/nfcapd
    .201201$(printf "%02d" $1)0000:nfcapd.201201$(printf "%02d" $1)2355
     -n 10 -s dstport $iplist -o csv > /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $1).csv
```

Listing B.4: Top10 ports based in IP-addresses

A script that uses the 10 most popular IP-adresses and their corresponding ports to find the number of flows sent to each port on each IP-address.

```
#!/bin/bash
for (( i = 1; i < 31; i++ )); do
  iplist=()
  ip=$(awk -F',' 'NR == 2 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $i).csv)
  iplist[0]=$ip
  ip2=$(awk -F',' 'NR == 3 { print $5}' /home/eeglarse/flowtest/top10/$
    (printf "%02d" $i).csv)
  iplist[1]=$ip2
  ip=$(awk -F',' 'NR == 4 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $i).csv)
  iplist[2]=$ip
  ip=$(awk -F',' 'NR == 5 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $i).csv)
  iplist[3]=$ip
  ip=$(awk -F',' 'NR == 6 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $i).csv)
  iplist[4]=$ip
  ip=$(awk -F',' 'NR == 7 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $i).csv)
  iplist[5]=$ip
  ip=$(awk -F',' 'NR == 8 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $i).csv)
  iplist[6]=$ip
  ip=$(awk -F',' 'NR == 9 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $i).csv)
  iplist[7]=$ip
```

```
  ip=$(awk -F',' 'NR == 10 { print $5}' /home/eeglarse/flowtest/top10/$
    (printf "%02d" $i).csv)
  iplist[8]=$ip
  ip=$(awk -F',' 'NR == 11 { print $5}' /home/eeglarse/flowtest/top10/$
    (printf "%02d" $i).csv)
  iplist[9]=$ip
  portlist=()
  ip=$(awk -F',' 'NR == 2 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[0]=$ip
  ip=$(awk -F',' 'NR == 3 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[1]=$ip
  ip=$(awk -F',' 'NR == 4 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[2]=$ip
  ip=$(awk -F',' 'NR == 5 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[3]=$ip
  ip=$(awk -F',' 'NR == 6 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[4]=$ip
  ip=$(awk -F',' 'NR == 7 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[5]=$ip
  ip=$(awk -F',' 'NR == 8 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[6]=$ip
  ip=$(awk -F',' 'NR == 9 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[7]=$ip
  ip=$(awk -F',' 'NR == 10 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[8]=$ip
  ip=$(awk -F',' 'NR == 11 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[9]=$ip
  for (( s = 0; s < 10; s++ )); do
      for (( j = 0; j < 10; j++ )); do
        $(nfdump -R /data/netflow/oslo_gw/2012/01/$(printf "%02d" $i)/
    nfcapd.201201$(printf "%02d" $i)0000:nfcapd.201201$(printf "%02d"
    $i)2355 -n 10 -s dstport -o csv 'dst ip ${iplist[$s]} and dst port
    ${portlist[$j]}' -o csv)
      done
  done
done
```

Listing B.5: Number of flows for each port and IP-address combination

A script extracting the values fetched in the preceding example, and creates .CSV-files for each IP-address and port combination.

```
#!/bin/bash
clock_converter(){
  if [ $(($1 % 100)) -gt 59 ]; then
    return $(($1 % 100))
  fi
  if [ $(($1 % 100)) -lt 60 ]; then
    echo $(printf "%04d" $1)
    return $(($1 % 100))
  fi
  }


for (( i = 1; i < 31; i++ )); do
  iplist=()
  ip=$(awk -F',' 'NR == 2 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $i).csv)
  iplist[0]=$ip
  ip2=$(awk -F',' 'NR == 3 { print $5}' /home/eeglarse/flowtest/top10/$
    (printf "%02d" $i).csv)
  iplist[1]=$ip2
  ip=$(awk -F',' 'NR == 4 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $i).csv)
  iplist[2]=$ip
  ip=$(awk -F',' 'NR == 5 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $i).csv)
  iplist[3]=$ip
  ip=$(awk -F',' 'NR == 6 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $i).csv)
  iplist[4]=$ip
  ip=$(awk -F',' 'NR == 7 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $i).csv)
  iplist[5]=$ip
  ip=$(awk -F',' 'NR == 8 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $i).csv)
  iplist[6]=$ip
  ip=$(awk -F',' 'NR == 9 { print $5}' /home/eeglarse/flowtest/top10/$(
    printf "%02d" $i).csv)
  iplist[7]=$ip
  ip=$(awk -F',' 'NR == 10 { print $5}' /home/eeglarse/flowtest/top10/$
    (printf "%02d" $i).csv)
  iplist[8]=$ip
  ip=$(awk -F',' 'NR == 11 { print $5}' /home/eeglarse/flowtest/top10/$
    (printf "%02d" $i).csv)
  iplist[9]=$ip
  portlist=()
  ip=$(awk -F',' 'NR == 2 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[0]=$ip
```

```
  ip=$(awk -F',' 'NR == 3 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[1]=$ip
  ip=$(awk -F',' 'NR == 4 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[2]=$ip
  ip=$(awk -F',' 'NR == 5 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[3]=$ip
  ip=$(awk -F',' 'NR == 6 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[4]=$ip
  ip=$(awk -F',' 'NR == 7 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[5]=$ip
  ip=$(awk -F',' 'NR == 8 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[6]=$ip
  ip=$(awk -F',' 'NR == 9 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[7]=$ip
  ip=$(awk -F',' 'NR == 10 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[8]=$ip
  ip=$(awk -F',' 'NR == 11 { print $5}' /home/eeglarse/flowtest/top10/
    top10port/$(printf "%02d" $i).csv)
  portlist[9]=$ip
  for (( s = 0; s < 10; s++ )); do
      for (( j = 0; j < 10; j++ )); do
        echo "date,value" >> daily_201201$(printf "%02d" $i)_${iplist[
    $s]}_${portlist[$j]}.csv
        for (( c=0; c<=2355; c += 5 ))do
          nfdump -r $(printf "%02d" $1)/nfcapd.201201$(printf "%02d" $i
    )$( clock_converter $c ) -n 10 -s srcip -o csv > /home/eeglarse/
    flowtest/2012/$(printf "%02d" $i)/$( clock_converter $c ).csv
          $total_file=$(awk -F',' 'NR == 15 { print $1}' /home/eeglarse
    /flowtest/2012/$(printf "%02d" $1)/$( clock_converter $c ).csv)
          echo $total_file >> daily_201201$(printf "%02d" $i)_${iplist[
    $s]}_${portlist[$j]}.csv
      done
        done
  done
done
```

Listing B.6: Create .csv files for each port and IP-address combination

Appendix C

Appendix C consist of the interview template for each interview, and the key results.

C.1 Template

The template is simply a facilitator to start the discussion with interviewee.

Part 1:
Explain the purpose of the interview, how the tools works, and what to look for.

Questions regarding nfdump:

1. In what degree do you use nfdump in your daily work?
2. What are the pros and cons of using nfdump?
3. How do you look for patterns in NetFlow packets today?
4. What would you like from a visual representation of NetFlow packets?

Demonstration of the visual solution.

Questions regarding potential in the solution:

5. How does the visual solution work in your opinion?
6. What is the potential in such a representation of the NetFlow packets?
7. What specific problems can such a solution solve?
8. In what level of detail should the visualization be?

C.2 Key results

Key results revolve around the potential in such a solution and what problems it might solve, e.g. question 6 and 7.
Focus on what purpose it would serve up against the current nfsen was how to quickly see changes is behavior as the possibility to see different representations of the data. It was quickly stated that such a solution could be used as a plug-in in a bigger tool. That it only paints a part of the bigger picture.

During the testing and answers in question 5 and 8 it was revealed that it served its purpose to find patterns and unusual behavior, but lacked details to go into more specific information, such as traffic between two individual IP-addresses and ports.