NTNU

Norwegian University of
Science and Technology

Management of Large Scale NetFlow Data by Distributed Systems

Zehua Tian

Master of Telematics - Communication Networks and Networked Services

Submission date: June 2016

Supervisor: Yuming Jiang, ITEM

Co-supervisor: Otto Wittner, ITEM
Arne Øslebø, UNINETT

Norwegian University of Science and Technology
Department of Telematics

Title: Management of Large Scale NetFlow Data by Distributed Systems

Student: Zehua Tian

Problem description:

UNINETT is the national research Internet Protocol (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 is an important technology available on most routers and switches. By analyzing NetFlow data, a picture of network traffic flow and volume can be built. As the number of running servers and routers at UNINETT has increased, managing the overall collection of NetFlow data collected has become a challenge. "Elasticsearch" and NoSQL databases are both popular distributed system techniques possible for dealing with such big data problem.

This thesis will investigate to what extent Elasticsearch and selected NoSQL databases can handle NetFlow data records, including exploring the capabilities of using them to manage large scale NetFlow data records for quickly searching, analyzing and troubleshooting network traffic, and comparing their efficiency. As looking up individual NetFlow entries has limited value, the systems' potential support for correlation and aggregation of flows is important. Novel techniques will be studied and suggested. Selected anomaly detection techniques may be relevant as test cases and will be studied and applied and tuned to the extent time allows.

Responsible professor: Yuming Jiang, NTNU

Supervisor: Otto Wittner, Arne Øslebø, UNINETT

Abstract

Nowadays, as network has almost permeated all aspects of people's life, network quality and security administration becomes very necessary. A very important part is monitoring and analyzing network traffic. NetFlow is an important technique for collecting network traffic information and it has been used extensively in network industry. As network keeps expanding rapidly both in size and complexity, management of collected large scale NetFlow data has met new challenges. New efficient tools are needed.

This thesis aims to investigate proper distributed NoSQL databases for handling large scale NetFlow data and mainly focus on their capabilities for quickly searching interesting information, analyzing and troubleshooting network traffic.

There are many different NoSQL databases, which can be broadly grouped into four types: key-value, column-family, document-oriented, graph-based. In this thesis work, the features and usages of different types of NoSQL databases are firstly studied. Based on that, the proper NoSQL database for this thesis are mainly selected through four aspects: data store, search ability, aggregation ability and extra useful features for data analysis. An integrated toolset: Elasticsearch, Logstash, Kibana (ELK) stands out to be a very promising solution. The three components of ELK work coordinately and can cover a complete NetFlow data analysis process from data collecting, store, process to visualization.

To further evaluate the capabilities and performance of selected ELK system, practical experiments of using ELK to manage real NetFlow data are carried out through three use cases: monitoring traffic statistics, surveying suspicious flows and detecting common attacks. The results show that the powerful searches and aggregations of Elasticsearch, advanced data pipeline of Logstash and rich visualizations of Kibana provide a very good solution. Some usage suggestions and further work are also discussed.

Preface

The following thesis is written for the degree of Master of Science at the Norwegian University of Science and Technology (NTNU). The thesis work is carried out at the department of Telematics, NTNU, Trondheim.

I would like to thank the supervisors Otto Wittner, Arne Øslebø, UNINETT and the responsible professor Yuming Jiang, NTNU for giving me insight, information, guidance and assistance. Otto Wittner and Arne Øslebø give me much help during the thesis project process including providing various resources, introducing technical knowledge, having many meetings with me and answering my questions. They give me important guidance for the project work. Professor Yuming Jiang supports me a lot in the thesis organizing and writing process with many good ideas and valuable advice. He always gave me very fast, useful and detailed feedback and which greatly improves the thesis. I sincerely appreciate them for this semester.

I also would like to thank all my families and friends for caring, chatting and having fun with me, which encouraged me and provided very helpful distractions when I encounter difficulties.

Contents

List of Figures

List of Tables

List of Source Code

List of Acronyms

API Application Programming Interface.

CSV Comma-separated values.

ELK Elasticsearch, Logstash, Kibana.

FTP File Transfer Protocol.

HTTP HyperText Transfer Protocol.

IETF Internet Engineering Task Force.

IP Internet Protocol.

IPFIX IP Flow Information Export.

JSON JavaScript Object Notation.

NTNU Norwegian University of Science and Technology.

SMTP Simple Mail Transfer Protocol.

SSH Secure Shell.

TCP Transmission Control Protocol.

ToS Type of service.

UDP User Datagram Protocol.

XML Extensible Markup Language.

Chapter 1
Introduction

1.1 Motivation and Objective

Nowadays, network has almost permeated all aspects of people's life. Network keeps expanding rapidly both in size and complexity. Monitoring and analyzing network traffic is a crucial part of network quality and security administration. The approaches to achieve this goal can be divided into two basic groups: inspection of full packet contents or only partial useful statistics describing network behaviors. Full packet capture approach can provide most insight into the network traffic, but it usually requires expensive hardware and substantial infrastructure for storage and analysis [40]. NetFlow is an important technique originally developed by Cisco and has been enabled on most routers and switches now. Compared with full packets, NetFlow data only contains many useful characteristic parameters of packets, which significantly reduces the amount of data to be analyzed and is more scalable for high-speed networks. It has been used extensively in network industry.

As network scale has grown greatly recent years, the amount of produced NetFlow data has become much bigger now. There are many good tools existing for NetFlow data analysis, but they have met new challenges when dealing with big NetFlow data. Additionally, the recorded network behaviors have also turned out to be more complicated and new aspects need to be looked into. New efficient management tools for large amounts of NetFlow data are necessary to be explored.

Distributed NoSQL databases are popular solutions for big data problems. This thesis aims to investigate proper NoSQL databases for handling large scale NetFlow data and mainly focus on their capabilities for quickly searching interesting information, analyzing and troubleshooting network traffic. Through this thesis work, the promising system should be chosen and the practical management capabilities should be explored.

1.2 Related Work

Through surveying, there are generally two kinds of previous work found about large scale NetFlow data and distributed systems:

- The first kind aims to solve the problem about efficiently collecting NetFlow data from a highly distributed network topology with a large amount of equipment. What are mostly concerned about NoSQL databases are their capabilities of efficiently writing and reading data with a distributed structure. NoSQL databases are mainly used as the data store and data provider. The analysis work is done by other third-party systems.
- The second kind builds complex distributed computing systems using Apache Hadoop to carry heavy and advanced MapReduce analysis on very large data sets.

They both are somewhat related to the thesis topic, but they are not what the thesis project mainly wants to deal with. As the thesis project is proposed by UNINETT, the distributed data collection aspect is not a problem for them. Also, Hadoop system is very powerful, but it's not the target for this thesis. NoSQL databases of their own may have good capabilities to analyze data. If they can be used both as data store and analysis tools, the whole management process will be greatly simplified.

In order to assist this thesis project, generally three kinds of work are studied:

- NetFlow data and current NetFlow data analysis tools, see more details in Chapter 2.
- NoSQL databases' features, see more details in Chapter 2 and Chapter 3.
- Network attack detection based on NetFlow data, see more details in Chapter 5.

1.3 Methodology

To achieve the thesis's goals, a survey of related and relevant techniques is followed by a tool selection phase, which again is followed by novel design of use cases and finally configuration and performance evaluation of selected tools applying the use cases.

Specifically, the thesis work is mainly carried out with two steps: NoSQL databases selection and the practical evaluation of the selected system for managing large scale NetFlow data. There are many popular NoSQL databases. They can be divided

into several types and the theoretical knowledge of each type are studied. In order to find the most suitable one for this thesis, firstly, the requirements for the desired new system are defined. Then based on the requirements, through investing and comparing different databases, the promising system is selected. For the selected candidate system, through working with a large amount of NetFlow data practically, novel use cases are explored and performance is tested.

1.4 The Structure of The Thesis

This thesis is divided into seven chapters in total, they are organized as the following structure:

- Chapter 1 gives a brief introduction to the thesis' motivation, objective, related work and methodology.
- Chapter 2 firstly introduces the background knowledge of NetFlow and the common "NFDUMP/Nfsen" toolset for dealing with NetFlow data. As NetFlow data becomes big data, the limitations of ordinary NetFlow data management tools are discussed and the popular big data solution - NoSQL databases are also introduced.
- Chapter 3 starts with defining the requirements for the desired new management system. Based on that, the functional features of different kinds of NoSQL databases are evaluated and the promising system is selected. For the selected system - ELK, technical features are studied in detail, which is the necessary theoretical preparations before starting practical work.
- Chapter 4 explains the experimental conditions and necessary setup procedures for running the selected system - ELK up to deal with NetFlow data.
- Chapter 5 presents novel usages of ELK for large scale NetFlow data management. Three main kinds of use cases are described in Section 5.2, Section 5.3, Section 5.4.
- Based on the findings of Chapter 5, Chapter 6 discusses two suggestions for using ELK.
- Chapter 7 contains a summary of the results for this thesis and discusses further work.

Chapter 2
Background

This chapter starts with introducing the general technical background knowledge about NetFlow, NetFlow data and NetFlow data based network analysis. After that, the current widely used methods for analyzing NetFlow data through NFDUMP and NfSen tools are presented. As network traffic grows increasingly in volume and complexity nowadays, the current approaches turn out to have several significant limitations. NoSQL databases provide solutions for big data management with good performances. Different types of NoSQL databases are also studied.

2.1 NetFlow

NetFlow is a network technology originally developed by Cisco for collecting IP traffic information [5], now widely available on most routers and switches. Compared with full packets capture, NetFlow only records network communication information through some useful network-layer and transport-layer attributes of packets. They can help gain insight into network behavior easily by providing information about who is using the network, when the network is utilized, what is the type of applications and how bandwidth is consumed.

2.1.1 NetFlow Data

NetFlow data usually calls flows. An IP flow is unidirectional and contains a group of packets sharing several common IP packet attributes [1]:

- Source and destination IP address
- Source and destination port
- Transport layer protocol
- Type of service (ToS)
- Router or switch input and output interfaces

These attributes identify unique or similar IP packets. They are examined by NetFlow-enabled routers or switches on each forwarded packet to decide if they belong to a new flow or should be added to an existing flow. Flows are constantly and temporarily filled into a local cache of routers or switches, called NetFlow cache. A single flow is ready for export when it is inactive for a certain time, lasts greater than the active timer or a Transmission Control Protocol (TCP) flag indicates the flow is terminated [1]. The whole process is depicted in Figure 2.1. Additionally,

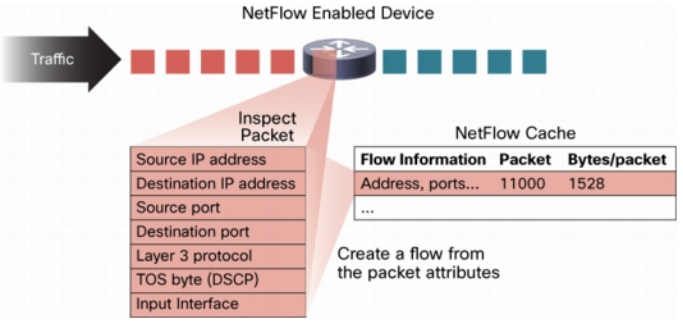


Figure 2.1: Creating a flow in the NetFlow cache [1]

there are also some other useful attributes describing more details about traffic, such as start and end time, lasting duration; TCP flags; number of packets and number of bytes. For each NeFlow data record(a flow), these attributes usually calls data fields.

There are several NetFlow exporting versions which define slightly different and evolved formats of flows over the years. Most important three among them are NetFlow v5, NetFlow v9 and IP Flow Information Export (IPFIX). NetFlow v5 is most commonly used, which contains fixed number of attribute fields in its record. It only supports IPv4. NetFlow v9 is extensible and contains much more information within a flow compared with NetFlow v5. A big change is that it is template-based allowing to define the record format flexibly. It means the data format can be adapted easily to provide support for new protocols. So NetFlow v9 is said to be "future-proofed" [2]. It supports IPv6 traffic. IPFIX is an Internet Engineering Task Force (IETF) standard based on NetFlow v9. It is a common, universal standard of flow formats [41].

2.1.2 NetFlow based Network Traffic Analysis

As introduced before, instead of containing actual payload of packets, NetFlow data only records useful attributes of Network communications. NetFlow based network traffic analysis is said to be easier and faster with less cost to realize since much smaller data volume is generated. Although NetFlow data reveals less information, they are still very useful to understand how the network is behaving with many

important use cases: network traffic monitoring, performance optimization, network accidents troubleshooting or anomaly detection and so on. They can be realized conveniently through various useful metrics or summary information derived from raw NetFlow data, e.g. draw bytes, packets volume diagrams over time; generate traffic distributions of protocols, IP addresses or ports; calculate bandwidth usage, maximum/average utilization and so on.

In order to apply NetFlow based network analysis, generally, there are three necessary components to be deployed in the system, which are NetFlow Agent, NetFlow Collector, and NetFlow Analyzer. The overview architecture is depicted in Figure 2.2.

Figure 2.2: The overview system architecture to work with NetFlow

This is just a logical diagram separating different parts based on the distinct functional roles. They could be software programs running totally or partially together on the same machines.

- NetFlow Agent usually are routers or switches that have the NetFlow feature enabled. They spread network and continuously generating flows. In practice, to prevent resulting in huge numbers of flow records which may consume too much bandwidth and disk space, usually not all the flows are captured. "Sampling" technique is configured for this. For example, a "1:N" sampling means only one packet will be examined out of every N packets. Sampling rate can be either fixed or random. Generated flows are exported periodically to NetFlow Collector.
- NetFlow Collector processes and stores flows collected from agents, and the NetFlow data can further be used by NetFlow Analyzer.
- NetFlow Analyzer has the abilities doing analysis work on flows either through searching interesting data, generating reporting information or visualization methods.

As the topic of this thesis is "management of NetFlow data", the management should at least include the functions of partial collector and analyzer.

2.2 Ordinary NetFlow Data Analysis Tools

NFDUMP is a set of command line based tools which can collect, store NetFlow data and also provide many powerful functions to analyze flows. NfSen is a web based front-end interface to display flows. They have been widely used together for analyzing NetFlow data over the years.

2.2.1 NFDUMP

NFDUMP contains a set of useful programs. Two most commonly used ones are nfcapd and nfdump. nfcapd is a NetFlow capture daemon which reads NetFlow data from the network and stores the data into files [46]. It works as a collector. Data is stored into a new file every fixed time period, which is configurable. Files store binary NetFlow data and use time-based names with the format of nfcapd.YYYYMMddhhmm. nfdump can read binary NetFlow data from the files produced by nfcapd and display netflow data information with several useful output formats. It also allows to process data for analyzing. The work process of them is depicted in Figure 2.3.

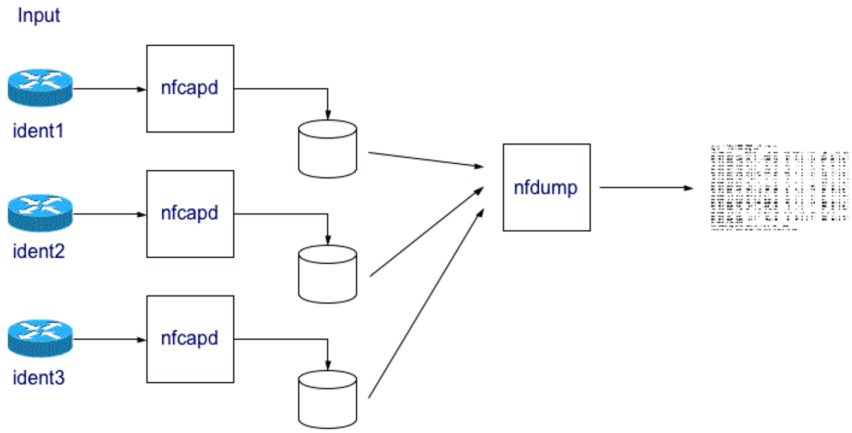


Figure 2.3: nfcapd and nfdump work process [46]

All data collected by nfcapd is stored as separated files to disk. To analyze data, nfdump firstly either reads a single file or a sequence of files at the same time and outputs them with friendly formats to help inspect data. For example, "read 10 minutes data between 2012.01.01 09:20 and 2012.01.01 09:30 and output data with the line format" uses command:

```
nfdump -R nfcapd.201201010920:nfcapd.201201010930 -o line
```

The result is shown in Figure 2.4.

```
Date first seen          Duration Proto      Src IP Addr:Port          Dst IP Addr:Port   Packets    Bytes Flows
2012-01-01 09:19:19.401     0.000 TCP    146.156.192.126:443   ->     158.27.92.20:37267         1     1402     1
2012-01-01 09:19:54.494     0.000 TCP      218.169.83.65:51860 ->   190.118.60.238:63569         1       40     1
2012-01-01 09:19:08.711     0.000 TCP     190.118.173.45:80    ->    94.201.72.91:62482         1       52     1
2012-01-01 09:19:53.988     0.000 TCP    161.222.137.109:50863 ->    74.200.34.104:80           1       40     1
2012-01-01 09:19:47.063     0.000 UDP     161.220.161.81:55055 ->    69.116.95.118:12536        1       73     1
2012-01-01 09:19:44.175     0.000 UDP     161.220.161.81:55055 ->    47.131.84.141:2933         1       73     1
```

Figure 2.4: nfdump line format output

The aggregation function of nfdump can aggregate flows according to the value of data fields. For example, through command:

```
nfdump -r nfcapd.201201010920 -A srcip,dstport
```

flows with the same source IP address and destination port are grouped together. Instead of looking at individual pieces of data, aggregations process data records and return computed results. The result is described in Figure 2.5. It is shown there are total 4 flows with source address 89.141.40.146 and destination port 8111.

```
Date first seen          Duration       Src IP Addr Dst Pt   Packets    Bytes      bps    Bpp Flows
2012-01-01 09:20:35.805     0.000   190.118.145.161  65531         1       58        0     58     1
2012-01-01 09:23:15.497     0.000     73.159.14.68     80         1       40        0     40     1
2012-01-01 09:21:20.490     0.000     68.244.224.78  37624         1       40        0     40     1
2012-01-01 09:22:25.441     0.000    191.220.45.180  14082         1       46        0     46     1
2012-01-01 09:23:04.022    41.711   190.118.162.163  53770         4     6000     1150   1500     1
2012-01-01 09:19:20.859   272.762     89.141.40.146   8111         5      200        5     40     4
```

Figure 2.5: nfdump aggregation output

Another useful function is the "TOP N statistics", which can have the N, a type and an order specified by users. For example, the command:

```
nfdump -r nfcapd.201201010920 -n 10 -s srcip/bytes
```

will generate top 10 source IP addresses which generate most bytes. The result is shown in Figure 2.6. This is also a kind of aggregations, which aggregates flows with same source IP addresses and further orders the results according to the total accumulated bytes.

nfdump also allows to retrieve flows selectively by defining custom filters. If filters are added into the command, all flows are filtered before they are further processed. Any filter consists of one or more expressions. Any number of expressions can be linked

```
Top 10 Src IP Addr ordered by bytes:
Date first seen          Duration Proto       Src IP Addr    Flows(%)     Packets(%)       Bytes(%)         pps      bps   bpp
2012-01-01 09:19:58.931   241.033 any      107.105.62.255       4( 0.0)   115441(65.8)    5.0 M( 9.5)      478   164756    43
2012-01-01 09:19:20.577   279.183 any      190.118.238.39      45( 0.2)     2566( 1.5)    3.6 M( 6.9)        9   103230  1403
2012-01-01 09:18:59.026   300.932 any      162.185.32.105     567( 2.3)     2618( 1.5)    3.6 M( 6.8)        8    94559  1358
2012-01-01 09:18:59.032   300.931 any     190.118.162.163     243( 1.0)     2014( 1.1)    3.0 M( 5.8)        6    79676  1488
2012-01-01 09:18:59.026   300.679 any      190.118.81.153      10( 0.0)     1460( 0.8)    2.2 M( 4.2)        4    58137  1496
2012-01-01 09:18:59.040   300.192 any     195.253.121.112      32( 0.1)     1233( 0.7)    1.7 M( 3.2)        4    44162  1344
2012-01-01 09:18:59.026   300.660 any     161.221.155.127      37( 0.2)      973( 0.6)    1.4 M( 2.8)        3    38417  1483
2012-01-01 09:18:59.030   300.659 any      161.221.159.22     312( 1.3)     1039( 0.6)    1.4 M( 2.6)        3    36407  1316
2012-01-01 09:18:59.239   300.108 any     191.220.195.154     280( 1.1)      941( 0.5)    1.3 M( 2.5)        3    35000  1395
2012-01-01 09:18:59.028   300.661 any      190.49.180.97      287( 1.2)     1095( 0.6)    1.3 M( 2.4)        3    33794  1159
```

Figure 2.6: nfdump top statistics output

together using "and", "or" and "not". For example "tcp and (src ip 107.105.62.255 or src ip 190.118.238.39) and not dst port 21".

NFDUMP functions both as NetFlow Collector and NetFlow Analyzer. It is widely used to process NetFlow data.

2.2.2 NfSen

NfSen is a web based front-end for nfdump to display flows graphically. It can use the power of nfdump as the back-end [47]. The overview integrated structure of them is shown in Figure 2.7.

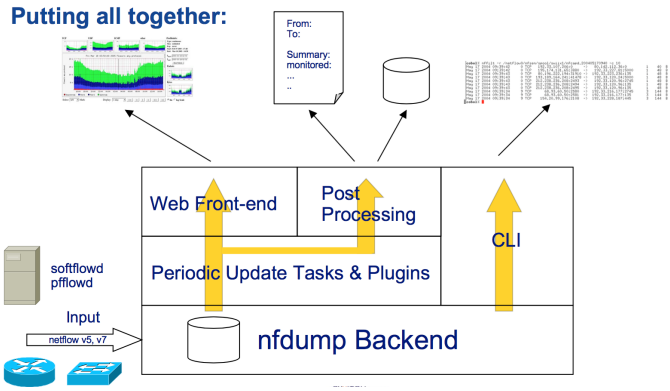


Figure 2.7: NFDUMP and NfSen toolset structure [39]

NfSen has many useful functions, including displaying flows, packets and bytes diagrams over time; easily navigating through and drilling down NetFlow data; filtering and analyzing NetFlow data using its web based as well as the command line based interface. Some examples are shown in Figure 2.8. The visualized approaches make NetFlow data analysis more efficient.

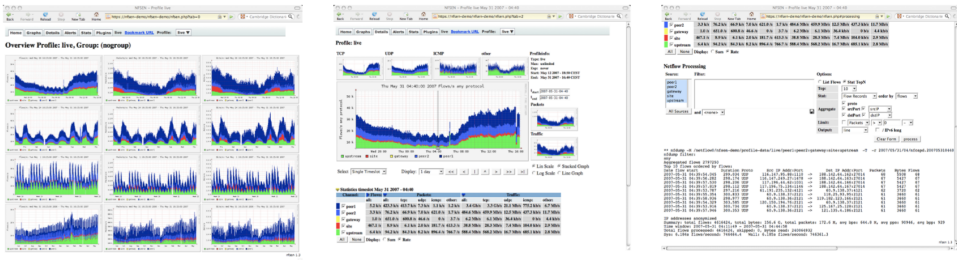


Figure 2.8: NfSen usage examples [47]

2.3 Management of Big NetFlow Data: NoSQL Databases

2.3.1 Introduction

Big Data

Along with the rapid development of network, big data arises naturally and rapidly in various forms: physical measurements, social network activities, commercial customer behaviors, system logs and so on. A common character they share is that single record of stored data does not generate much value while there exist important patterns and trends in the data which can help with many aspects like quality evaluation, decision making or even useful prediction. But it is usually not fully known where to look or how to find them in advance. Additional software techniques are needed to manage big data.

Big NetFlow Data

UNINETT develops and operates the Norwegian national research and education network [3]. As network complexity of UNINETT keeps growing over the years, the number of running servers and routers in UNINETT has increased a lot. The volume of produced NetFlow data has also become much bigger now. Figure 2.9 is the map of current UNINETT network. As seen from it, the UNINETT network's gateways spread most places all over the Norway. They all keep producing new NetFlow data all the time. It is said UNINETT interconnects about 200 Norwegian educational and research institutions and more than 300 000 users, as well as giving them access to international research networks [3]. In order to gain useful insight into this kind of network, managing the overall collection of NetFlow data with current tools has met some challenges because of limited processing capabilities.

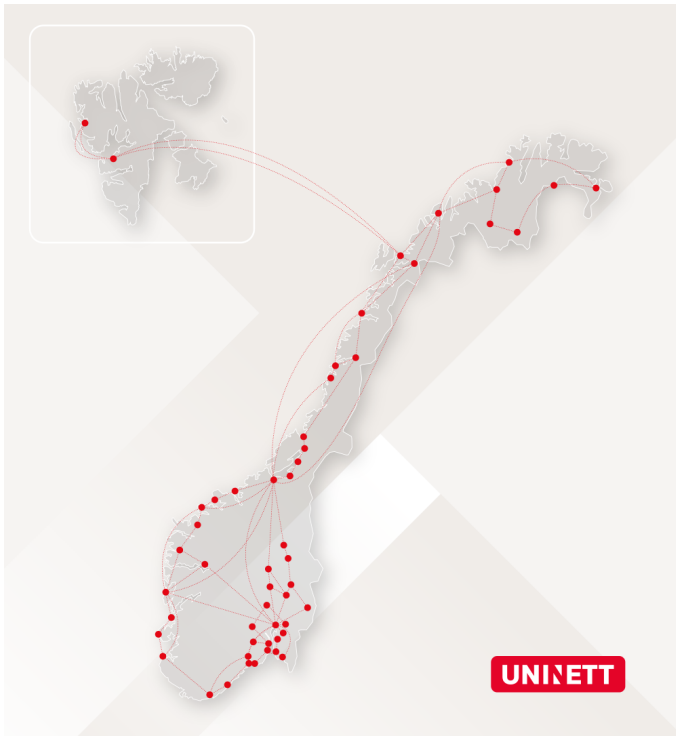


Figure 2.9: Map of the UNINETT network [4]

2.3.2 Limitations of Ordinary Tools

As introduced before, NFDUMP and NfSen toolset is very useful for NetFlow data management. But when encountering large scale NetFlow data all over the complex distributed Network, ordinary tools like NFDUMP have some significant limitations:

- Inefficient file based data store. Usually NFDUMP stores NetFlow data into a new file every five minutes, which will result in a large number of files. Analyzing data needs to manually pick up data by specifying the paths and names of files every time. If NetFlow data files are stored at different places, they need to be transferred and centralized on the same machine in order to get the overview analysis results. These operations are not difficult, but they will be very cumbersome and not flexible when needing to be constantly repeated.
- Very slow processing speed. The analysis of a large amount of NetFlow data through NFDUMP will take a lot of time because NFDUMP always reads data from each file line by line from the beginning. Although just for executing a simple command to pick up a small amount of interesting data, if data spreads

many files over a large volume of data, every single line of those files all need to be read by NFDUMP which obviously will be very slow to get the desired result.

– Limited analysis methods. NFDUMP does provide some useful functions for analyzing NetFlow data. As the network situation is becoming more and more complex nowadays, new aspects of NetFlow data need to be researched which requires new analysis approaches.

Above all, when NetFlow data becomes big data and records more sophisticated network situation, new tools and methods are needed. NoSQL databases are popular solutions for big data problems. This thesis wants to explore the capabilities of NoSQL databases for the management of big NetFlow data.

2.3.3 NoSQL Databases

NoSQL Introduction

Generally, big data technique is challenged in three ways: the amount of data (volume), the rate of data generation and transmission (velocity), and the types of structured and unstructured data (variety) [49]. NoSQL(popularly translated as "not only sql") databases provide solutions for big data management with good performances. Compared with traditional relation databases, they are relatively new and tend to be open-source and have distributed architecture and flexible data schema[45].

Distributed architecture means data can be spread horizontally across many servers. Through native mechanisms of NoSQL databases, data and operations load can be automatically balanced across different servers, and when a server goes down, it can be quickly and transparently replaced without application disruption [37]. Traditional databases usually run on a single server. A reason of this is their different data models.

Data records in traditional databases is stored in rows and different tables store data records with different types (columns). Data in different tables is often related and needs to be queried combined. NoSQL databases are aggregate oriented, which means having a fundamental unit of storage, which is a rich structure of closely related data [28]. As all the data units are treated equally in this kind of design, it is easy to assign them to different servers automatically. This difference is illustrated in the Figure 2.10 with an example of recording books information. The two kinds of databases are designed to fit different purposes with their own benefits. NoSQL databases are the right choice for big data.

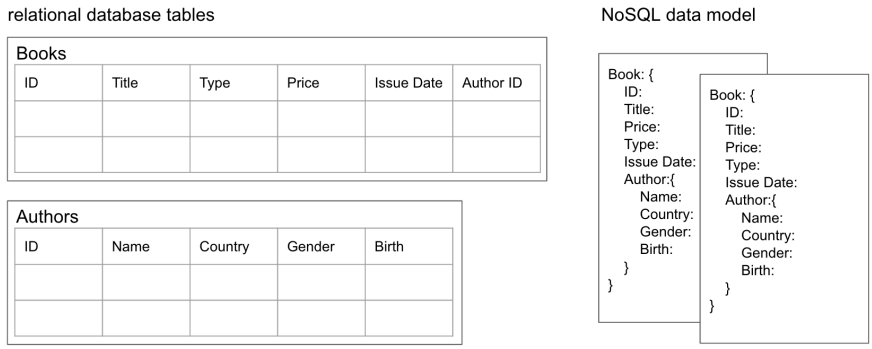


Figure 2.10: Tabluar and aggregated data model

Types of NoSQL Databases

Unlike common tabular data store in traditional relational databases, NoSQL databases realize flexible structures of data store. They can broadly be categorized into four types [38]:

- **Key-Value**
  Key-Value is usually considered the most basic type of NoSQL databases from the data model perspective. Each item in this kind of database consists of a pair of unique key and the matched value. There is no type or structure limitation to the value which could be either a simple numerical value or an object with nested structure inside. Some practical database realizations also allow to define the data structure for the value part, e.g. string, list or hash. The database operations are usually very simple: get the value for a key, put a value for a key, or delete a key.

  The key based operations can be very fast. But with less concerns to the value part, this kind of database has its specific use cases. It fits very well for quickly storing and accessing a large number of data pieces and no need to search on the specific content of data, e.g. website sessions management.

- **Column-Family**
  Column family can be treated as an advanced key-value edition. Now each key represents a row while each value is a set of columns with the column name and value. This maybe sound like a table of traditional relational database. But essentially, they are quite different. A first difference easy to see is no predefined structure of table with fixed columns is required. Each row still represents a single record. But various rows don't have to consist of same columns. Each column is also a key-value pair consisting of a column name and its value.

Columns of related data are grouped into column families in this kind of database. A column family is how the data is stored on the disk [43]. All the rows in a single column family is stored together which suits one or several columns of data are frequently to be read at once. It is not necessary to define columns for a column family and rows in a column family can have different columns.

The kind of database is good at storing very large numbers of records with large amounts of sparse data. Compared with traditional relational database, it can save a lot of storage and a query can target interested columns of data conveniently.

– **Document-Oriented**
The unit of data records in this kind of database is called a document. A document also looks like a key-value from the outermost. The value part is a structured set of key-value pairs (which can be called different fields) encoded in e.g. Extensible Markup Language (XML), JavaScript Object Notation (JSON). Nested fields are allowed, so a document can have complex hierarchical tree structures. It is not necessary to pre-define document structures and documents may have different structures. A document equals to the object concept in many programming languages, which is close to the real life data model. It is popular used in web applications.

Due to the flexibility of documents schema, this kind of database can manage a large variety of data records that differ in structure.

– **Graph-Based**
This is a quite different kind of databases. Information in Graph-Based database is represented using nodes, edges and properties. Nodes are data objects. Edges and properties define relationships. Edges connect different nodes with directions. Properties belong to edges defining the types of relationships. So data is organized by relationships which allows to find interesting patterns of data.

Graph Databases can be very powerful when data is highly connected and related in some way. Social network data is a popular use case.

Chapter 3
Management of Large Scale NetFlow Data: The ELK Approach

As discussed in the former chapter, NetFlow data is crucial for network analysis and nowadays, management of big NetFlow data needs more competent methods: NoSQL databases. There are many different NoSQL databases. This chapter firstly presents the requirements for the desired new management system. Then based on the new requirements, through looking into more function details of NoSQL databases, the proper candidate is selected. ELK turns out to be a promising solution for this thesis project. For each component of ELK : Elasticsearch, Logstash and Kibana , more technical details are introduced, which are helpful to form an overview idea before the practical experiments.

3.1 Requirements for the New Management System

Before starting to survey the proper NoSQL database solutions for this thesis, it is necessary to make some criteria. The desired requirements for the new management systems are summarized below:

- More flexible and convenient methods for organizing collected NetFlow data, e.g. storing data and deleting aged data.
- Advanced search and aggregations capabilities for analyzing big NetFlow data efficiently.
- Other more useful features for managing NetFlow data, e.g. visualization, simplicity and so on.

3.2 NoSQL Databases Selection

There are many different NoSQL databases according to this NoSQL databases list[1]. It is a key step to choose the proper one that can address the requirements both efficiently and sustainably. The selection is carried out among popular NoSQL

[1] http://nosql-database.org/

databases listed in the DB-Engines Ranking[2] with two steps in this thesis: primary selection and further selection. The primary selection is based on the first two requirements. In order to decide the most promising system, the selected candidates from this step will be further researched based on the third requirement.

3.2.1 Primary Selection

Matching the first two requirements for the desired new system, the primary selection investigates three main aspects: data store, search and aggregation features.

Data Store
The purpose of this thesis is to manage NetFlow data. NetFlow data consists of flows and each flow record contains different data fields. From the introduction in Section 2.3.3, graph-based databases have their specific use purpose, which is to store data with obvious relationships. So they are not the right choice for the thesis project. The other three types all can store data records with many fields. But key-value databases are designed for key based fast accessing a large amount of data and care less about the data value. Obviously, key-value databases are not meant for complex queries attempting to connect multiple pieces of data. When using them away from their design purposes, it is likely to exhibit poor performance and may need much unnecessary extra work to tune them. Column-family and document-store types are evolved forms of key-value store and they concern more on the data value. Obviously, they are more suitable for the thesis's purpose.

Search Ability
In order to analyze NetFlow data, it is a frequent procedure to search based on the value of a single or combinations of fields. For example, "get all the TCP flows of a specific source IP address within some time range" involves "protocol", "source IP address" and "timestamp" fields. In order to execute this kind of searches and more importantly, get the results quickly, all data fields need to be indexed to become searchable.

An index is any data structure that improves the performance of lookup [51]. Generally, indexing a data field builds an inverted structure which allows to use the value of the fields to find the records containing it. Otherwise, databases must scan all the data records to select those matching a query statement. It is just like what nfdump does and it is highly inefficient. Through researching several popular NoSQL databases, it is found indexing data fields are supported in many column-family and document-oriented databases. It can be realized through settings or commands both during or after importing data by specifying column names or field names which are desired to be searchable.

[2] http://db-engines.com/en/ranking

Aggregation
The next important requirement is the "aggregation" capabilities. When analyzing big NetFlow data, aggregated reporting information is usually more interested than separate data records. It is better to use a practical aggregation example for explaining, e.g. "get the top 10 source IP addresses which generated the most flows within a time range". This is a common and useful aggregation in network traffic analysis. Through surveying several popular document-oriented and column-family NoSQL databases, it is found aggregation function is generally implemented in two ways: MapReduce and native aggregation framework.

MapReduce is a programming model to summarize and run aggregation functions on large data sets across many servers. Data is processed on each server in parallel firstly and then all the results are combinned into one set. A MapReduce aggregation is composed of a Map() method that performs filtering and sorting (such as in the example above, sorting flows by source IP addresses into queues, one queue for each unique source IP addresses) and a Reduce() method that performs a summary operation (such as counting the number in each queue) [52]. It is found only a few NoSQL databases realize embedded MapReduce framework while most NoSQL databases need extra supports from "Apache Hadoop" [50] software framework. Deploying Hadoop with NoSQL databases will add much extra complexity to the data management process while embedded MapReduce realization provides more simplicity. So only embedded MapReduce framework is considered.

While MapReduce needs to define custom functions to perform the map and reduce operations, native aggregation framework provides a rich set of frequent aggregation functions directly, which are easier to understand and use. Aggregation framework operation is also usually faster than MapReduce because of the fact that aggregations happen in memory. In general, MapReduce can provide more advanced aggregation results through custom functions, but it is less efficient and more complex than native aggregation framework.

Through researching several popular document-oriented and column-family databases, two of them stand out because of good aggregation capabilities: Elasticsearch and MongoDB. Other NoSQL databases either may need extra Hadoop support or have limited aggregation functions compared with these two ones. Elasticsearch provides many useful aggregation functions in its aggregations framework while MongoDB has both a native aggregation framework and embedded MapReduce framework. They both seem preliminary fulfill the requirements of data store, advanced search and aggregation requirements for managing big NetFlow data. More details and extra features of them need to be looked into for further selections.

3.2.2 Further Selection

Elasticsearch is more essentially an open source search engine built on top of Apache Lucene that delivers a full-featured search experience across terabytes of data [6]. It is also usually treated as a NoSQL database since it has the general database functions. MongoDB is a leading open-source NoSQL databases widely used in web applications. They are both classified as document-oriented NoSQL databases. According to the survey before, they both seem satisfy the basic requirements of this thesis project. To further decide between them, more extra features are evaluated on them.

As introduced in Section 2.2, the graphical tool NfSen obviously enriches the power of NFDUMP for analyzing NetFlow data. Visualizations based analysis approaches are very helpful for understanding a large amount of data quickly. The possibilities of the new system to work with visualizations approaches conveniently are explored.

It is found there is a specific web front-end tool - Kibana developed for Elasticsearch by the same company. MongoDB also has a visualization tool called Compass developed by the same company. But the visualization usages in Compass and Kibana are different in fact.

Compass uses visualizations to help understand data in the database (e.g. data type, data value and data structure) and build sophisticated queries through the graphical user interface operations. This kind of visualization is more about how data resides in databases and help with operating on data. But Kibana works more than that. What makes Kibana really powerful for aiding in data analysis is that it allows to draw various diagrams to visualize the results of Elasticsearch aggregations. For example, Top N aggregations can be either expressed as a table to show the result items or a pie chart to show more clear about the percentage.

What is more, it is found there is a novel popular toolset called ELK starting to be used more and more for data analysis. ELK is a toolset consisting of Elasticsearch, Logstash and Kibana. They are three separate open source software. Logstash works as collecting and importing data to Elasticsearch. The combination of them covers the functions from collecting data, storing data to analyzing data. They are all developed by the same company and are open-source software. ELK can make the whole large scale NetFlow data management process much simpler and smoother. The overview working process of them is depicted in Figure 3.1:

Above all, ELK seems a good choice for this thesis project and is decided as the first candidate. It will be studied in more details and practically experimented firstly. MongoDB is saved as the second candidate.

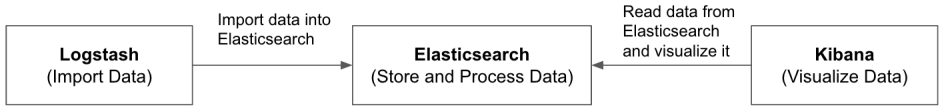


Figure 3.1: The explanation of ELK.

3.3 The Selected System: ELK

This chapter presents the components of ELK: Elasticsearch, Logstash and Kibana in more details about their concepts, syntax and functions, which are helpful to form an overview idea before using it practically. Elasticsearch can be sophisticated when deep into and also has many powerful capabilities for full-text search. For this thesis, only related features will be introduced.

3.3.1 Elasticsearch Distributed System

Elasticsearch is designed to be distributed. It can scale out with many servers and handle a large amount of data. To better understand how Elasticsearch works in a distributed structure, knowing the units at different levels of the whole system and how they work together can help draw a clear picture quickly.

Basic concepts [16]

- **Cluster:** A cluster is the outermost container in the distributed structure which may consist of one or many servers. Each cluster has a unique name. It is responsible for organizing different servers to work coordinately, e.g. store data and search data spreading them.
- **Node:** A single server is called a node with a node name. When a node is started, it will join the cluster with the desired name. In a new system, starting a single node will by default form a new single-node cluster named "elasticsearch".
- **Index:** An index is a collection of data records that have similar characteristics and are logically grouped together. Many indexes can be defined in a single cluster. An index is identified by a name.
- **Type:** Inside an index, different types can be defined to further category data.
- **Document:** The basic data record unit is called a document. A document is expressed in the JSON format and usually have different data fields.

Table 3.1: Concepts comparison between Elasticsearch and relational databases

Relational Database	Elasticsearch
database	index
table	type
row	document
column	field

– **Shard:** Each index is subdivided into multiple pieces called shards when storing on disk. Each shard can be hosted on any node in the cluster. There are two kinds of shards: primary shard and replica shard. Primary shards store data while replica shards are the copies of primary shards allocated to a different node which help provide high availability.

To help understand them, they can be matched with their similar concepts in traditional relation database as shown in Table 3.1. Cluster, node and shard are the new concepts in the distributed system.

Work Mechanism [34]
Inside a cluster, nodes form a full mesh topology, which means that each Elasticsearch node maintains a connection to each of the other nodes. A master node is selected automatically, which is in charge of managing cluster-wide changes and can be replaced automatically if fails. When starting a new node in a new system, a new cluster with the node is created and the node becomes the master node. As depicted in Figure 3.2.

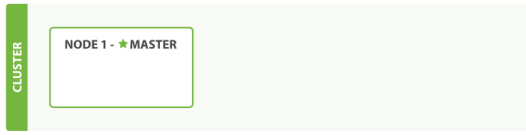


Figure 3.2: A cluster with one empty node [31]

After creating a new index and importing data to it, several primary shards are created. The number of both primary and replica shards are configurable. As replica shards need to locate at a different node from the node of primary shards. So no replica shards are created in this single node cluster yet. As depicted in Figure 3.3.

When adding a new node, the replica shards are also created on the different nodes. As depicted in Figure 3.4.

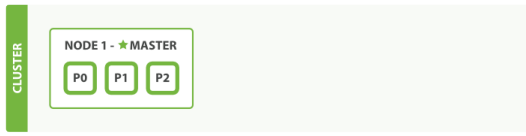


Figure 3.3: A single-node cluster with an index [29]

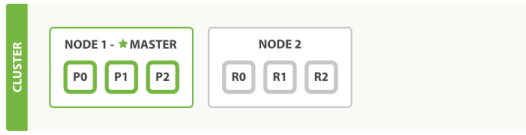


Figure 3.4: A two-node cluster-all primary and replica shards are allocated [30]

When more nodes are added, shards are reallocated to spread the load. As depicted in Figure 3.5.

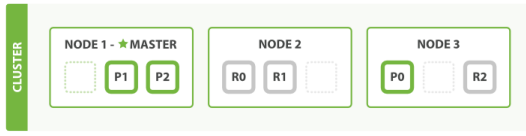


Figure 3.5: A three-node cluster-shards are reallocated to spread the load [35]

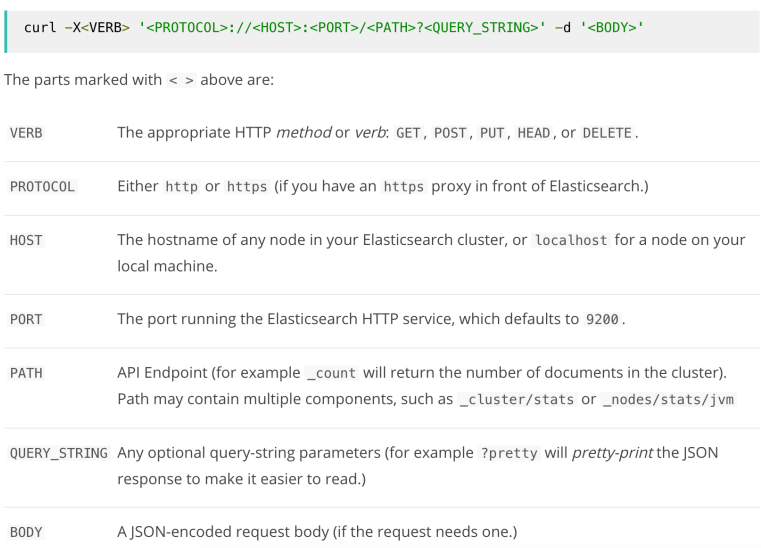
Elasticsearch can also adjust the distribution when nodes fail automatically. As depicted in Figure 3.6.

Figure 3.6: Cluster after killing one node [32]

The distributed design makes Elasticsearch manage a large amount of data by using the storage and processing capabilities of many servers parallel and also provide high availability.

3.3.2 Talk to Elasticsearch

Elasticsearch can be communicated using a RESTful Application Programming Interface (API) over HyperText Transfer Protocol (HTTP) with a specified port number (9200 by default), either through command line or several programming languages. The easiest way is using the "curl" command, which is an open source command line tool for transferring data with URL syntax. The common structure of requests and each part are depicted in Figure 3.7:



```
curl -X<VERB> '<PROTOCOL>://<HOST>:<PORT>/<PATH>?<QUERY_STRING>' -d '<BODY>'
```

The parts marked with `< >` above are:

`VERB`	The appropriate HTTP *method* or *verb*: `GET`, `POST`, `PUT`, `HEAD`, or `DELETE`.
`PROTOCOL`	Either `http` or `https` (if you have an `https` proxy in front of Elasticsearch.)
`HOST`	The hostname of any node in your Elasticsearch cluster, or `localhost` for a node on your local machine.
`PORT`	The port running the Elasticsearch HTTP service, which defaults to `9200`.
`PATH`	API Endpoint (for example `_count` will return the number of documents in the cluster). Path may contain multiple components, such as `_cluster/stats` or `_nodes/stats/jvm`
`QUERY_STRING`	Any optional query-string parameters (for example `?pretty` will *pretty-print* the JSON response to make it easier to read.)
`BODY`	A JSON-encoded request body (if the request needs one.)

Figure 3.7: Explanation of an Elasticsearch request [36]

Different HTTP methods in VERB part are used for different purposes, like creating, deleting or getting data from index/indexes. The "PATH" part indicates both the target indexes or types and API commands of different purposes provided by Elasticsearch, e.g. "_search" for executing a search or aggregation request or "_settings" for retrieving settings of index/indexes.

For example, a request to count the number of documents in the cluster would be:

Source Code 3.1 Count the number of documents in the cluster.

```
curl -XGET 'localhost:9200/_count?pretty' -d '
{
    "query": {
        "match_all": {}
    }
}'
```

3.3.3 Search in Elasticsearch

It is very easy to build complicated and robust queries in Elasticsearch. Elasticsearch provides a rich and flexible query language called the query DSL(domain-specific language), which uses a JSON request body. Search body is included within "query" parameter. There are generally two kinds of queries: Leaf query and Compound query.

Leaf queries are single purpose queries used by themselves. There are two subset of leaf queries: full text queries and term level queries. Only term level queries are used in this thesis. Term level means treat the value of a data field as whole while full text further divides text value. The full introduction of them sees Term level queries [22]. For example, "terms" query filters documents that have fields that match any of the provided terms, as shown in Source Code 3.2.

Source Code 3.2 Terms query example.

```
curl -XGET 'localhost:9200/_search?pretty' -d '
{
    "query": {
        "terms": {
         "pr":["TCP", "UDP"]
        }
    }
}'
```

"range" query matches documents with fields that have terms within a certain range, as shown in Source Code 3.3.

Source Code 3.3 Range query example

```
curl -XGET 'localhost:9200/_search?pretty' -d '
{
    "query": {
        "range": {
         "ts":{"from" : "2012-01-01 22:00:00", to : "2012-01-01 23:00:00"}
        }
    }
}'
```

Compound queries wrap other compound or leaf queries. The full introduction of them see Compound queries [10]. For example, "bool" query selects documents matching boolean combinations of other queries, as shown in Source Code 3.4, different parts(should, must_not, filter) are optional.

3.3.4 Aggregation in Elasticsearch

The aggregations framework in Elasticsearch provides very useful aggregated information. Aggregations usually operate on the results of the search parts. Both search

Source Code 3.4 Bool query example

```
curl -XGET localhost:9200/_search?pretty -d '
{
    "query": {
        "bool": {
             "should": [
                  { "term": { "da": "192.168.1.1" }},
                  { "term": { "da": "192.172.22.12" }}

             ],
             "must_not": { "range" : { "sa" : { "gte" : "161.223.0.0", "lte" : "161.223.255.255"} }},
             "filter": [
                  { "range" : { "sp": {"gt": "1023"}} },
                  { "term" : { "pr": "TCP" }}
             ]
        }
    }
}
```

and aggregation are included in a single request. There are two important basic concepts for mastering aggregations [33]:

- Buckets: Collections of documents that meet a criterion.
- Metrics: Statistics calculated on the documents in a buckets.

Each aggregation is simply a combination of one or more buckets and zero or more metrics. What makes the aggregations of Elasticsearch really powerful is that aggregations can be nested. A top-level aggregation executes within the context of the executed search request. The sub-aggregations will be computed for the buckets which their parent aggregation generates. The structure of aggregations is shown in Figure 3.8.

```
"aggregations" : {
    "<aggregation_name>" : {
        "<aggregation_type>" : {
            <aggregation_body>
        }
        [,"meta" : {  [<meta_data_body>] } ]?
        [,"aggregations" : { [<sub_aggregation>]+ } ]?
    }
    [,"<aggregation_name_2>" : { ... } ]*
}
```

Figure 3.8: The basic structure of aggregations [7]

There are many different types of aggregations in Elasticsearch. They can be broken into three main families [7]:

- Bucketing: This kind of aggregations build buckets which means select documents according to criteria and group them into relevant buckets. The result will be a list of buckets - each one with a set of documents that "belong" to it. For example all the TCP flows with a same source IP address can be grouped into a bucket.
- Metric: It refers to the aggregations that keep track and compute metrics over a set of documents. They can be nested into bucketing aggregations.
- Pipeline: These aggregations works on the output of other aggregations and their associated metrics.

Specific aggregation types will be introduced when used practically later. A complex aggregation example could be Source Code 3.5. This request gets top 10 sources IP addresses which send most bytes and calculate the most number of bytes sent by a single source IP address.

Source Code 3.5 A complex aggregation example

```
curl -XGET localhost:9200/_search?pretty -d '
{
  "query": {
    "term" : { "pr" : "TCP" }
  },
  "aggs" : {
    "TOP_N_TCP_IP" : {
      "terms" : {
        "field" : "sa",
        "size" : "10",
        "order": { "BYTES": "desc" }
      },
      "aggs" : {
        "BYTES" : {
          "sum": { "field": "byt" }
        }
      }
    },
    "MAX_BYTES" : {
      "max_bucket": {
        "buckets_path": "TOP_N_TCP_IP>BYTES"
      }
    }
  }
}'
```

"TOP_N_TCP_IP" , "BYTES" and "MAX_BYTES" are three custom aggregations names. "TOP_N_TCP_IP" is a bucket aggregation which group data according to the same source address. "BYTES" is a metric aggregation and is nested inside

"TOP_N_TCP_IP". For each source address, it will add all the bytes of its flows together. "MAX_BYTES" is a pipeline aggregation which select the maximum value of "BYTES" aggregation. The other words are all the key words and their value.

3.3.5 Logstash

Logstash is an open source data collection engine. The generic use purpose of it is for receiving data, transforming it, and outputting it. To be more specific, it can unify data from different data sources, cleanse and normalize collected data for the chosen destination outputs [26]. It works like a pipeline for advanced transporting data.

From the technical implementation perspective, Logstash is a collection of: Input, Filter, Output and Codec plugins. The input plugins consume data from a source, the optional filter plugins modify the data, and the output plugins write the data to a destination. Codec plugins are predefined representations of some common data sources, e.g. Apache system log data, Nmap data, and they are used as part of input or output plugins to filter desired data. The working mechanism is depicted in Figure 3.9.

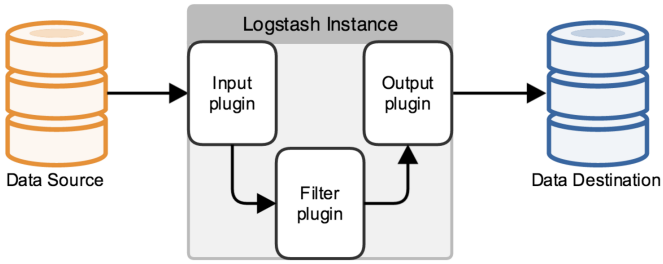


Figure 3.9: The structure of Logstash pipeline [27]

3.3.6 Kibana

Kibana is an open source browser-based interface designed to work with Elasticsearch. It provides friendly graphical interfaces to search, view, and interact with data stored in Elasticsearch indexes [24], which greatly enrich the power of Elasticsearch through visualizations. There are four function taps available in the Kibana web interface. "Discover" tab displays Kibana's data discovery functions and "Setting" tab is used for configuring settings. "Visualize" and "Dashboard" are the two most frequently used taps which provide various visualization functions. As shown in Figure 3.10, "Visualize" tap provides many options of different diagrams. Essentially, these diagrams visualize the results of bucket aggregations and metric aggregations generated from Elasticsearch.

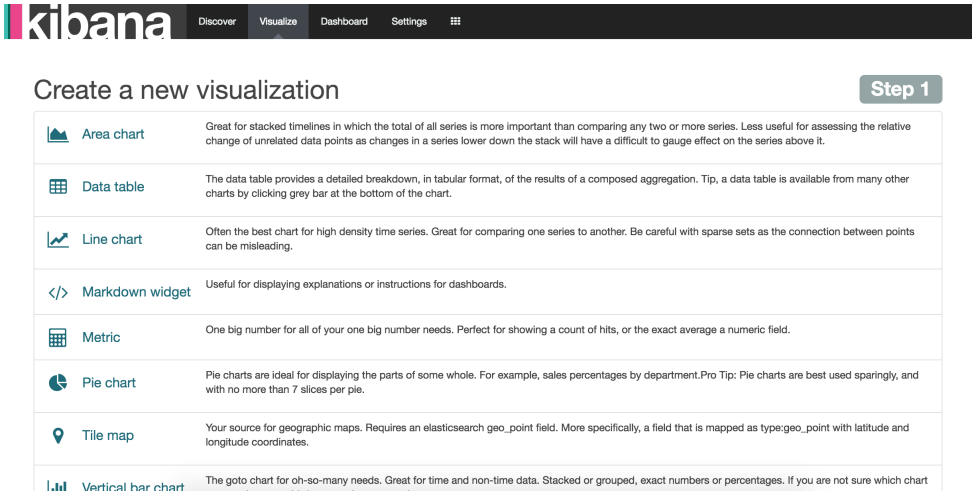


Figure 3.10: Kibana "Visualize" interface

For example, after clicking on the "Pie chart", the interface for making a pie chart is shown, as depicted in Figure 3.11. Usually for a same set of data, many different diagrams can be made, which can help better understand data. "Dashboard" allows to load a collection of diagrams together and arrange them flexibly through simple drag and drop.

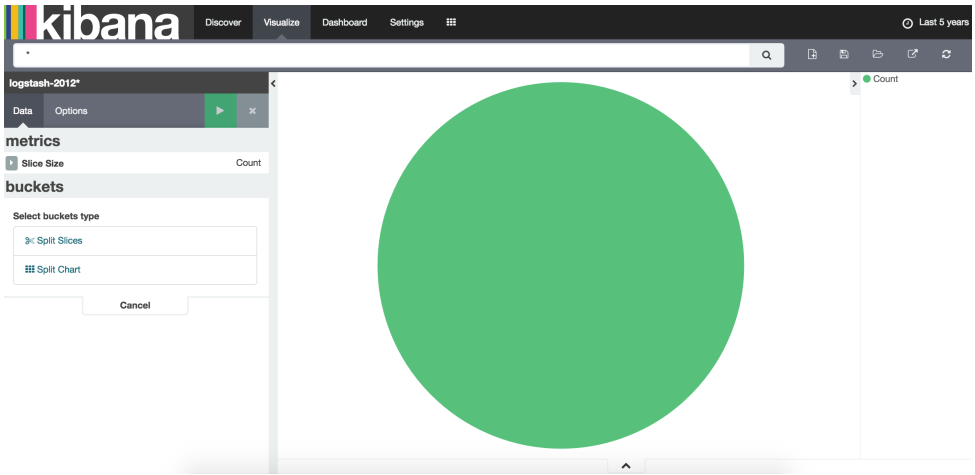


Figure 3.11: Kibana "Pie Chart" interface

3.3.7 Other Useful Integreated Tools

Except Elasticsearch, the company which built it also develops several other open source software to fully help make sense of data. They altogether are designed to take data from any source and search, analyze, and visualize it in real time. Elasticsearch can be integrated to work together with them naturally, which provides more comprehensive power for dealing with data.

Except Logstash and Kibana, there are also several other useful tools for strengthening more aspects of Elasticsearch, like Shield (security), Watcher (alerting), and Marvel (monitoring). They all together greatly extend Elasticsearch usages for dealing with data.

Chapter 4
System Setup

After the former theoretical investigations and preparations, ELK has been selected as the system for the management of large scale NetFlow data. At the next stage, practical evaluations are necessary. In this chapter, the details for preparing the ELK system to be ready for analyzing NetFlow data are explained.

4.1 Data Set and Equipment

NetFlow data analysis could both be real-time or historical. In this thesis project, historical NetFlow data in 2012 is provided by UNINETT in the form of nfcapd capture files. A single file contains data of five minutes and "1:1000" sampling was applied.

The main equipment of the experimental work in this thesis are the "iou2" apache server at UNINETT with Ubuntu operating system and a personal computer which can login to the server remotely using Secure Shell (SSH).

4.2 ELK Setup

For simplicity, Elasticsearch, Logstash and Kibana are all installed on the "iou2" server for the experimental work. Practically, they could distribute on different servers according to the whole production deployment and performance requirements. In this case, the Elasticsearch cluster only has one node.

Installation:
Elasticsearch and Logstash require Java. A recent version of Oracle Java should have been installed on the server. It is very easy to install Elasticsearch, Logstash and Kibana by downloading and unzip the latest releases of them from the company website[1]. Alternatively, they could be installed from the RPM or Debian repositories.

[1] https://www.elastic.co/

In this thesis, the former method is used. The commands for installing them through command line is shown below:

```
curl -O  "URL of ZIP file"
tar zxvf
```

Configuration [11]:
ELK have proper default settings for starting to try them out. But there are several settings are necessary or better to be tuned before they are used practically according to some performance findings of the experiments in this thesis project:

- Because of the fact that aggregations happen in memory, applying aggregations on data fields requires all the value of those fields to be loaded into memory. When the size of loaded data exceed the allocated memory, "out of memory" error will occur. There are two most useful settings better adjusted for Elasticsearch when working on a large amount of data for frequent and complex aggregations.

  The first one is "ES_HEAP_SIZE" environment variable which allows to set the heap memory that will be allocated to Elasticsearch java process. The default value is "1g" which turns out to be too small for big data set. Before modifying this value, when applying nested aggregations, "out of memory" errors happen frequently. The value of "20g" is set in this thesis experimentally and greatly improves the performance of aggregations. The setting command is shown below:

  ```
  export ES\_HEAP\_SIZE=20g
  ```

  The second one is "mlockall" which tries to lock the process address space into memory, preventing any Elasticsearch memory from being swapped out by operating systems. This can be done by adding:

  ```
  bootstrap.mlockall: true
  ```

  to the "elasticsearch.yml" file. This file is the main configuration file of Elasticsearch. There are also many other settings[2], which may also need to be adjusted when necessary.

- As introduced before, Logstash works with a set of input, filter and output plugins. To run Logstash, the specific definitions of them are needed. Usually the information is provided in the form of a configuration file. Details about this configuration file will be explained in Section 4.3.2.

[2]https://www.elastic.co/guide/en/elasticsearch/reference/current/setup-configuration.html

– Kibana works as a graphical web front-end based on the Elasticsearch data back-end. It gets data from Elasticsearch and visualizes it. So Kibana needs to know the running address of Elasticsearch. By default, it will connect Elasticsearch from localhost with port number 9200. In "kibana.yml" file, it should have the setting like:

```
elasticsearch_url: "http://localhost:9200"
```

If Elasticsearch is configured differently, this setting needs to be changed accordingly.

Running:
To run Elasticsearch and Kibana, in the command line, change to the installation directory path and use command:

```
bin/elasticsearch or bin/kibana
```

Logstash needs to specify the configuration file when running, using command:

```
bin/logstash -f path/logstash.conf
```

Architecture:
The overview architecture of the whole system is depicted in the Figure 4.1. By default, Kibana listens on “localhost” with port number 5601. In order to visit the Kibana web interface remotely from the browser on the personal computer, an apache proxy is configured on the "iou2" server, which can forward all the public requests to the port 8080 of "iou2" server to the Kibana application. This could be achieved through installing "mod_proxy" module of apache server and configure a "VirtualHost *:8080" for Kibana [44]. For security considerations, to only allow legal users to access Kibana from this public address, a username and a password are set. The installation, configuration and running of ELK are operated on the personal computer through connecting to "iou2" server via SSH.

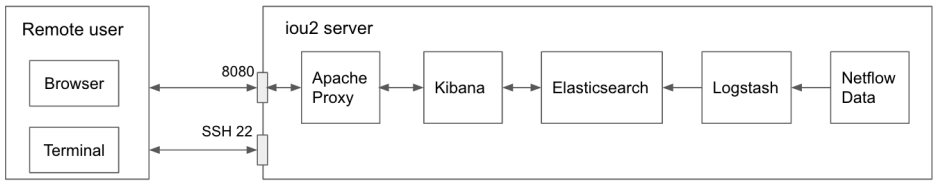


Figure 4.1: The architecture of the experimental ELK system.

4.3 Importing NetFlow Data

In order to use Elasticsearch to deal with NetFlow data, data needs to be imported into Elasticsearch first. To import data properly, three following steps explained in the following three subchapters are necessary.

4.3.1 NetFlow Data Preparation

The available historical NetFlow data is stored in the binary nfcapd capture files, which can't be used as data input for Logstash directly. It is necessary to transform the data into a format that Logstash can recognize. One solution is that nfdump can read data from capture files and output data as Comma-separated values (CSV) format conveniently. In this format, each flow occupies one line with value of different fields separated by comma. Many programs can read this kind of data easily including Logstash. The command for this is:

```
nfdump -r "netflow capture file path" -o csv > "output file path"
```

From the output result, it is shown a complete flow record used in this thesis has 39 fields in total. But many fields have zero value which seems meaningless. After confirming with UNINETT, the reason is that not all fields are configured for their capture. Obviously, for a large amount of data, these useless fields will waste much storage and affect the performance to deal with flows in ELK later. It is better to delete these fields from output results. Except several native output formats, nfdump allows to define customer output formats flexibly with a format description "fmt:<format>". Different fields have predefined element tags in nfdump. For the experimental work of this thesis, ten fields are selected for each flow record as shown in the Table 4.1:

The nfdump command for converting is:

```
nfdump -r "file path" -o "fmt:%ts %td %pr %sa %da %sp %dp %pkt %byt %flg" -q > "output path"
```

Table 4.1: Selected NetFlow data fields

Tag	Description	Tag	Description
%ts	Start Time - first seen	%sp	Source Port
%td	Duration	%dp	Destination Port
%pr	Protocol	%pkt	Packets
%sa	Source Address	%byt	Bytes
%da	Destination Address	%flg	TCP Flags

The "-q" option is for suppressing an unnecessary header line and some statistics at the bottom, which can't be recognized by Logstash. Since there are large numbers of files. Linux script is used in this thesis to execute converting automatically. For example, to convert all the data files in Jan 2012, the Linux shell script is shown below:

```
for (( i= 1; i <= 31; i++ ))
do
  cd in_path/2012/01/$i/
  for file in ./*
  do
    nfdump -r "$file" -o "fmt:%ts %td %pr %sa %da %sp %dp %pkt %byt %flg %fl" -q > out_path/201201$i/"$file"
    echo "$file finish"
  done
  echo "201201$i finish"
done
```

4.3.2 Logstash Configuration

After the former preparation, data for the input to Logstash has been ready. As mentioned before, when using Logstash, usually a configuration file needs to be provided. A Logstash configuration file mainly consists of three sections. Each section is explained separately below:

Input: in order to read NetFlow data from files, the "file" plugin is used. It also quite easy to use other proper input plugins for collecting real-time data or data from some other sources. The input part is shown below:

```
input {
  file{
    path => ["/data/netflow/20120115/nfcapd.20120116*"]
    start_position => "beginning"
    ignore_older => 0
    type => "netflow"
    sincedb_path => "/data/zehuat/null"
  }
}
```

"path" use files' paths array as value and filename patterns are allowed to match multiple files. "start_position" chooses where Logstash starts initially reading files. Since historical data is used in this thesis, "ignore_older" is set false to allow to read old files. The "type" of imported data is defined as "netflow".

Filter: as shown in Figure 4.2, the value of different data fields in one record of NetFlow data prepared before is separated by unfixed number of spaces because nfdump wants to print it beautifully. To get the seperate value of data fields successfully, a very powerful filter plugin "grok" is used. Grok works by defining grok patterns with the syntax "%SYNTAX:SEMANTIC" to match input data: the

```
2012-01-16 15:53:52.517     0.000 TCP      190.119.161.4  159.152.145.176  57522     80        1       40 .A....
2012-01-16 15:54:33.681     0.000 TCP      190.119.161.4  159.152.145.176  55244     80        1       40 .A....
2012-01-16 15:54:48.424     0.000 TCP      194.29.96.219    161.223.1.108  41990    443        1       64 .A....
2012-01-16 15:54:45.380     0.000 TCP       81.182.157.4    162.185.32.85  57341     80        1       40 .A....
2012-01-16 15:54:06.716     0.000 TCP    144.102.136.251   161.222.82.149  56323  20745        1       48 ....S.
2012-01-16 15:53:58.773     0.000 TCP      190.49.117.91   213.152.80.187   1548     80        1       40 .A....
```

Figure 4.2: Converted NetFlow data format.

SYNTAX is the name of the pattern that will match the interesting text and the SEMANTIC is the regex identifier given to the piece of text being matched [25]. Logstash provides many native patterns and allows to create custom patterns as well. The filer used in thesis is depicted below:

```
filter {
  grok {
    patterns_dir => ["./patterns"]
    match => {
        "message" => "%{TS:ts}%{SPACE}%{DURATION:td}%{SPACE}%{WORD:pr}%{SPACE}%{IPV4:sa}
        %{SPACE}%{IPV4:da}%{SPACE}%{PORT:sp}(?:\.[0-9]+)?%{SPACE}%{PORT:dp}(?:\.[0-9]+)?%{SPACE}
        %{NUMBER:pkt}%{SPACE}%{NUMBER:byt}%{SPACE}%{FLAG:flg}%{SPACE}%{NUMBER:fl}"
    }
    remove_field => ["message", "path", "host"]
  }
}
```

Custom patterns are written into a file and the path is set through "patterns_dir" parameter. "remove_field" parameter can remove some unnecessary meta data fields added by Logstash.

Output: "elasticsearch" output plugin is used for sending NetFlow data to Elasticsearch.

```
output {
  elasticsearch {
    hosts => ["localhost:9200"]
    index => "NetFlow-20120116"
    action => "index"
  }
}
```

"hosts" sets the connection address of Elasticsearch. "index" defines index name. "action" could be any of index, delete, create or update according to the purpose.

4.3.3 Elasticsearch Mapping

When data is sent to Elasticsearch, mappings tell Elasticsearch how to deal with imported data fields, e.g the types of value, index it or not. By default, native

"dynamic mapping" will do this automatically, which tries to guess the data type according to the value. But it is better to define custom mappings explicitly according to the specific imported data since it is more reliable and controllable. The mapping request of this thesis is described in Source Code 4.1.

Source Code 4.1 Create an Elasticsearch Mapping

```
curl -XPUT http://localhost:9200/_template/NetFlow_entry -d '
{
    "template" : "NetFlow-2012*",
    "mappings" : {
      "netflow" : {
        "properties": {
          "ts":{"type":"date", "format": "yyyy-MM-dd HH:mm:ss.SSS"},
          "td":{"type":"float"},
          "sa":{"type":"ip"},
          "da":{"type":"ip"},
          "sp":{"type":"integer"},
          "dp":{"type":"integer"},
          "pr":{"type":"string", "index" : "not_analyzed"},
          "flg":{"type":"string", "index" : "not_analyzed"},
          "pkt":{"type":"long"},
          "byt":{"type":"long"},
          "fl":{"type":"integer"}
        }
      }
    }
}'
```

"template" define settings and mappings that will be used for a set of new indexes. In the example below, the "NetFlow-entry" template will be applied for all the indexes with the name pattern "NetFlow-2012*". As an index may have several types, each type can has its own mapping. "netflow" is the type set for the data in this thesis. In the "properties" part, for "string" fields, "not_analyzed" means they won't be additionally processed for full-text search.

Chapter 5
Data Analysis using ELK

This chapter presents the practical usages for ELK to handle large scale NetFlow data. It starts with discussing how data is organized in Elasticsearch. After that, three kinds of use cases are mainly experimented: monitoring traffic statistics, surveying suspicious flows and detecting common attacks. Novel methods for analyzing NetFlow data by using ELK are explored and the performance is evaluated and discussed.

In the practical work of this thesis, two weeks' data from January 2012 of Oslo gateway will be mainly used for the demonstration. The second week is used to support the experimental findings of the first week. The dates of data are listed in Table 5.1.

5.1 A Novel Data Indexing Plan

One of the first decisions to make when using ELK to start to analyze practical large scale NetFlow data is how to structure data store. At UNINETT, NetFlow data is captured every 5 minutes which could last for many days, weeks and even several years. Data needs to be sent to Elasticsearch and indexed before operating on them. Simply importing all the data as a whole block into a single index has significant drawbacks:

– Performance aspect. For NetFlow data, like other time-based data streams such as system logs, social-network activities, new data is produced continually in nature. So the number of documents in the index grows rapidly accelerating

Table 5.1: Experimental data

Mon	Tue	Wed	Thu	Fri	Sat	Sun
16	17	18	19	20	21	22
23	24	25	26	27	28	29

with time. Although usually only the data within a some short time period is interested, searches and aggregations need to be applied on the whole huge index, which is obviously not efficient.

– Management aspect. Historic NetFlow documents are almost never updated, and analysis mostly targets the most recent documents. As documents age, they lose value and needed to be deleted to reduce the disk usage. New documents are created all the time and maybe need some modifications, e.g. a new data field, or a different data type. A single index is not flexible for deleting or updating operations.

There is no rule that limits to using only a single index. Time-based Elasticsearch indexes turn out to be a good choice for managing large scale NetFlow data. This means that for every day, a new index is created and all data for that day is stored within that index. The index name takes the format "NetFlow-YYYYMMDD". This approach can solve the drawbacks above easily:

– An Elasticsearch search request allows to target multiple indexes at the same time. So It's easy to query over only the days wanted. For example, the request:

```
curl -XGET localhost:9200/NetFlow-20120116, NetFlow-20120118/NetFlow/_search
```

searches within two indexes containing two days' data. Elasticsearch also supports wildcards matching for the index name – the query to "NetFlow-201201*" will be executed over all the data produced in January. Besides, another good feature "date math" [13] is supported in the index name. For example, a date math name template can restrict the search to the past two day by expressing index name as "<logstash-now/d-1dYYYY.MM.dd>".

– When wanting to delete unnecessary data, only related indexes of related days need to be deleted. If it is needed to configure a new mapping adding new data fields or updating types and settings, only new indexes should be matched and there is no need to worry about already imported indexes.

5.2 Use Case 1: Mornitoring Traffic Statistics

For stable network, traffic should follow some regular patterns, usually without strange peak behaviors or sudden changes. Through monitoring traffic statistics, abnormal behaviors could be noticed easily. This subchapter will investigate how ELK can be used for monitoring traffic and troubleshooting abnormal network behaviors.

Network traffic can be monitored per day and per week separately. Thirteen kinds of useful information are selected for each time unit, which could be divided into four categories according to the methods used for retrieving them from raw NetFlow data through Elasticsearch:

- TOP N statistics through "Terms aggregation"[23]. It is a bucketing aggregation which builds buckets for each unique value of the desired data field. The order of results and the returned number of buckets can be specified.
- Sum statistics through "Sum aggregation"[21].
- Traffic over time statistics through "Date Histogram aggregation"[12]. Through specifying a time interval and a timestamp data field, this aggregation can build a bucket every the time interval orderly.
- Unique count through "Cardinality aggregation"[9].

The results of various Elasticsearch aggregations can be visualized conveniently in Kibana. All the diagrams of those thirteen kinds of information can be grouped together shown in a single dashboard. So monitoring network traffic just needs to watch the various statistics in a single dashboard.

5.2.1 Daily Statistics

The daily dashboard of 16th is shown in Figure 5.1, which consists of: the ring diagrams of top 20 source/destination IP addresses/ports; the line diagram of bytes sum over time; the vertical bar diagram of flows sum over time; the table diagram of protocols; the metric diagrams of total flows/bytes sum and unique source/destination IP addresses/ports counts.

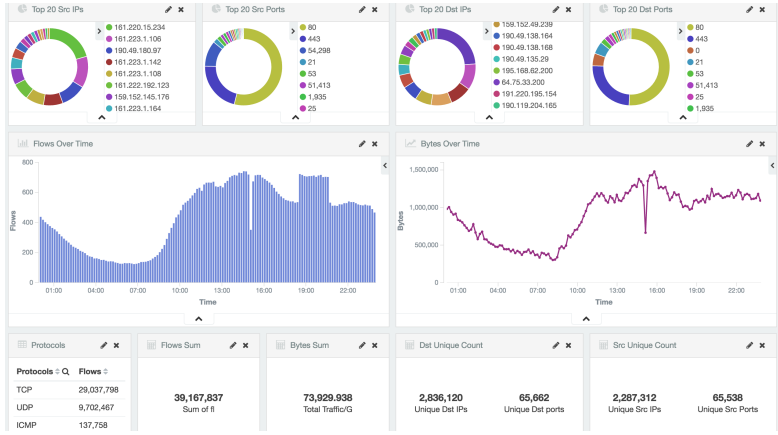


Figure 5.1: Daily traffic statistics dashboard.

From "Bytes Over Time" and "Flows Over Time" diagrams on the daily dashboard, it is easy to notice two significant abnormal time slots on 16th:

– "15:00 - 15:10" with sudden big decrease of both total bytes and flow counts, named as "Peak Decrease".

– "18:30 - 20:40" with sudden big increase of flow counts, named as "Peak Increase".

Without even more background knowledge of that day, through the next steps, it is proved ELK can help diagnose such abnormal network situations efficiently.

– **Step 1:** Since different protocols usually have different practical use purposes, network traffic should be analyzed separately. It is really easy to realize this through Kibana, after clicking on "TCP" row in the "Protocols" table diagram, a "TCP" filter is added to the Elasticsearch requests for regenerating the whole dashboard. The "TCP" and "UDP" traffic dashboards are shown in Figure 5.2 and Figure 5.3. It is easy to see that all the TCP traffic only has the "Peak Decrease" while all the User Datagram Protocol (UDP) traffic at that day has both "Peak Decrease" and "Peak Increase".

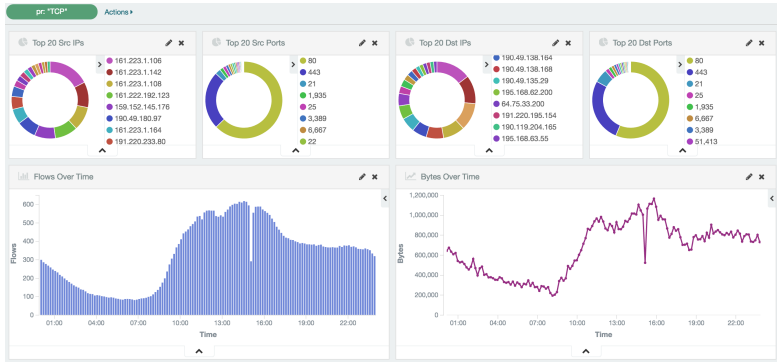


Figure 5.2: Daily TCP traffic statistics dashboard.

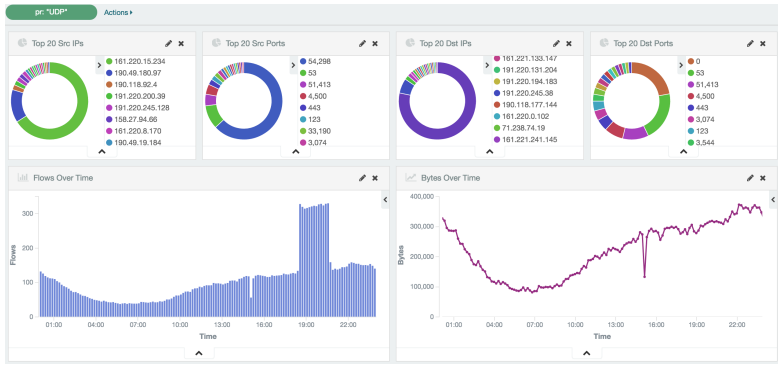


Figure 5.3: Daily UDP traffic statistics dashboard.

– **Step 2:** Then start to research the "Peak Decrease" in TCP traffic first. It is better to narrow down the time range to get more detailed views. After updating the time range filter to the "15:00-15:10" time slot and also setting a smaller time granularity of 5 seconds to time series' diagrams, as shown in Figure 5.4, there is almost no traffic at all during 15:04:20 til 15:08:20. Through further verification, the same result can be found on all the traffic of all the other protocols. According to the confirmation with UNINETT, this could be a sudden link down due to power off or equipment restart/repair.

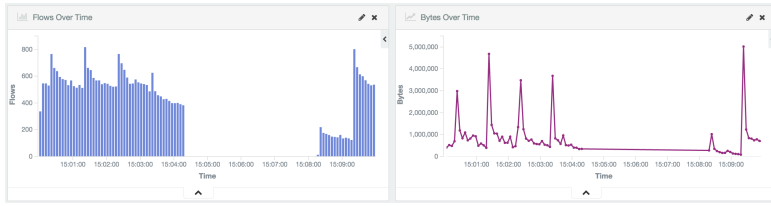


Figure 5.4: TCP suspicious traffic.

– **Step 3:** Next look into the "Peak Increase" in UDP traffic. As shown in Figure 5.5, after narrowing down the time range, the sudden increase of flows shows more clear. What is more, as seen from the "Top 20 Src IPs" and "Top 20 Dst IPs" ring diagrams, it is very suspicious that both source address "161.220.15.234" and destination address "79.63.46.204" generated almost 90% flows at that time period.

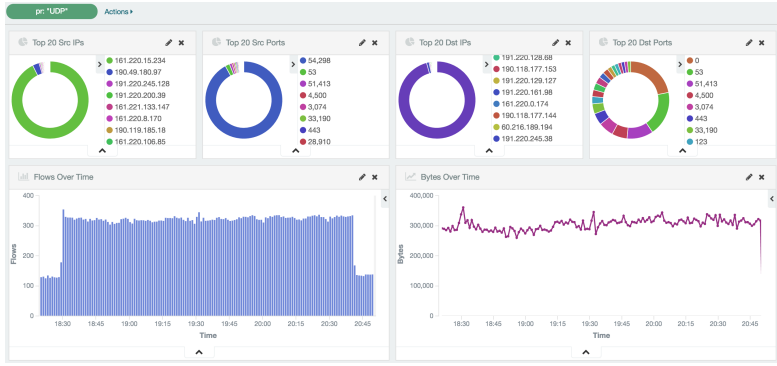


Figure 5.5: UDP suspicious traffic.

After clicking on the green ring part in the "Top 20 Src IPs" diagram, a new filter to watch only the traffic with source address "161.220.15.234" is added to the dashboard. From the Figure 5.6, it is found this source address only talked with destination address "79.63.46.204". And this conversation only existed

within that abnormal time slot.

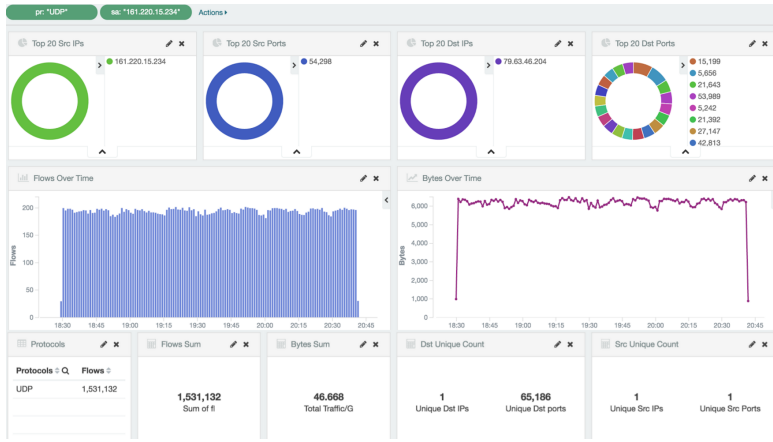


Figure 5.6: UDP suspicious traffic of source IP 161.220.15.234.

Further, it can be verified by inverting the source address filter to only watch traffic without source address "161.220.15.234". As shown in Figure 5.7, "Peak Increase" is removed. So the conversions between source address "161.220.15.234" and destination address "79.63.46.204" are the reason of "Peak Increase".

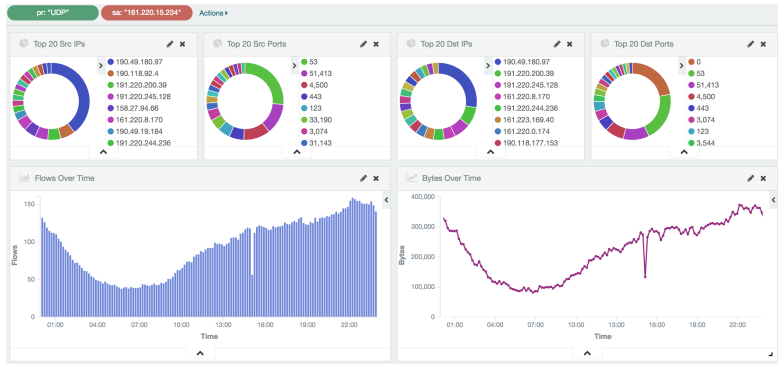


Figure 5.7: UDP traffic without source IP 161.220.15.234.

– **Step 4:** From the former steps, it has been found the "Peak Increase" is caused by the traffic between source address "161.220.15.234" and destination address "79.63.46.204". As shown in the former Figure 5.6, there is an abnormal increase period of flows lasting between "18:30 - 20:40" while not so much increase of bytes and there is only one unique source port while there are 65186

destination ports. After confirming with UNINETT, it could be a large scale UDP port scan for detecting services.

5.2.2 Weekly Statistics

The weekly traffic dashboard is similar to the daily dashboard with some adaptive modifications. In Kibana, there are limitations of the number of buckets in time series' diagrams in case too many points generated when visualizing. So the time granularity is set as hourly now. As the methods for analyzing flows of different protocols are similar, for the simplicity of demonstration, the following parts of the thesis will mainly focus on TCP flows. As seen from the Figure 5.8, it is also easy to notice several suspicious time slots.

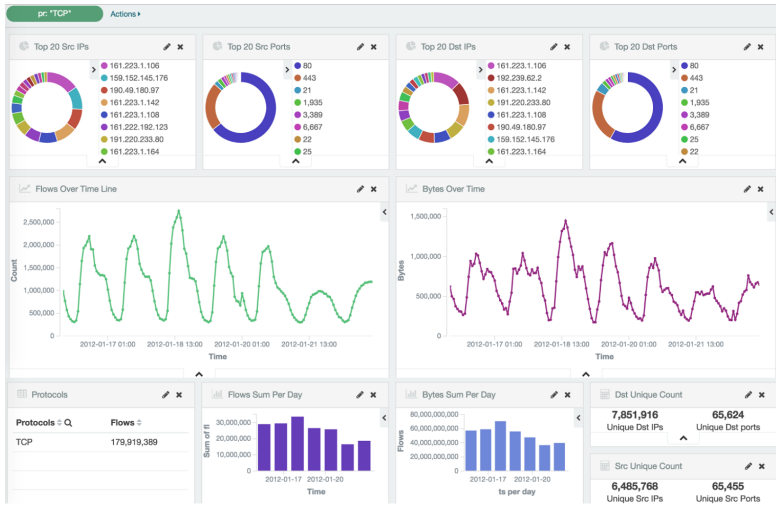


Figure 5.8: Weekly TCP traffic statistics dashboard.

The most obvious one is that the time slot "10:00 - 20:00" on 18th has a significant increase on both flows number and bytes compared with the other days of that week. This is even more obvious when drawing the flows diagram for two weeks' data, as depicted in Figure 5.9.

Through many experimental attempts, it is turned out that the reason can't be directly derived using the direct visualization methods used before, which simply rely on finding obvious characteristics in visualized diagrams. It is because when watching the daily dashboard of 18th, as seen from the Figure 5.10, there are no obvious peak behaviors at that single day in fact. It is a peak behavior reflected among several days. In order to diagnose this, efficient methods are needed to compare the traffic between different days in this case. The next subchapter will try to deal with it.

Figure 5.9: Flows over time diagram of two weeks' TCP traffic.

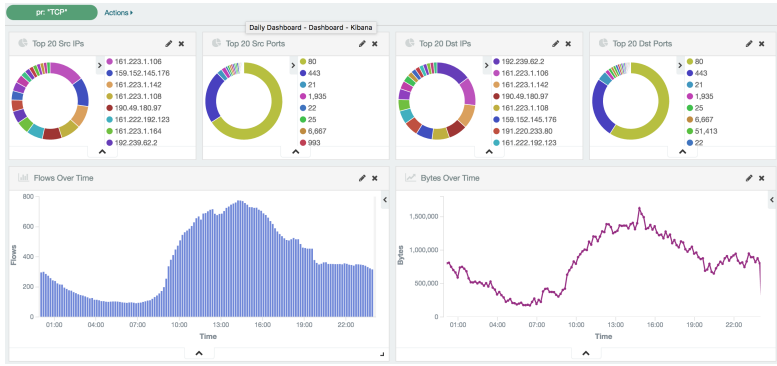


Figure 5.10: Daily traffic statistics of 2012.01.18 .

5.3 Use Case 2: Surveying Suspicious Flows

5.3.1 Filtering Suspicious Flows

As mentioned before, it needs to compare traffic between different days to diagnose the big traffic increase on 18th. Through several experiments, two methods are designed to achieve this goal and are introduced in the thesis. Essentially, the idea is still similar as before: first narrow down to abnormal time periods and then try to find abnormal IP addresses.

Top Statistics

The idea is that abnormal traffic must be caused by the abnormal behaviors of some IP addresses. Especially, the top ranked IP addresses have significant influences. Normal network traffic should have an average pattern for top ranks everyday. When

comparing the top ranks at the abnormal day with an average top ranks, some abnormal IP addresses could be filtered out. To verify this idea, within the suspicious time slot 10:00-20:00, the top 20 ranks of 18th and the average ranks of the rest days (16th, 17th, 19th, 20th) within the same week are compared. These top ranks can be generated easily through "Terms Aggregation"[23] applied on the search results with time range filters. The Elasticsearch request for getting the average top 10 rank is in Source Code 5.1. As weekend traffic has obvious difference with workday's, 21th(Saturday) and 22nd(Wednesday) are not included.

Source Code 5.1 Average top 20 TCP source IP addresses.

```
curl -XGET localhost:9200/logstash-20120116,logstash-20120117,logstash-20120119,logstash-20120120/
netflow/_search?pretty  -d '
{
    "size" : 0,
    "query": {
        "bool": {
            "should":[
                { "range" : { "ts": {"gte": "2012-01-16 10:00:00.000", "lte": "2012-01-16 20:00:00.000"}} },
                { "range" : { "ts": {"gte": "2012-01-17 10:00:00.000", "lte": "2012-01-17 20:00:00.000"}} },
                { "range" : { "ts": {"gte": "2012-01-19 10:00:00.000", "lte": "2012-01-19 20:00:00.000"}} },
                { "range" : { "ts": {"gte": "2012-01-20 10:00:00.000", "lte": "2012-01-20 20:00:00.000"}} }
            ],
            "minimum_should_match" : 1,
            "filter": [
                { "terms" : { "pr" : ["TCP"]} }

            ]
        }
    },
    "aggs" : {
          "Top_N_TCP_IP" : {
              "terms" : {
                  "field":"sa",
                  "size":"20"
              },
              "aggs": {
                  "FLOWS_COUNT": {
                      "value_count": { "field": "da" }

                  },
                  "AVG_FLOWS_COUNT" : {
                      "bucket_script": {
                          "buckets_path": {
                              "FLOWS_COUNT": "FLOWS_COUNT"
                          },
                          "script": "FLOWS_COUNT/4"
                      }
                  }
              }
          }
    }
}'
```

Two criteria are applied for deciding suspicious IP addresses: a. Rank rise; b. Flows increase. In this thesis experiments, if an IP address's rank increases equal or more than 3 or if the amount of increased flows is large enough relative to the average amount(e.g with same magnitude), the IP address is thought to be suspicious. For

example, through comparing the weekly average top 20 source IP addresses and those on 18th, the selected suspicious IP addresses are summarized in Figure 5.11, marked as yellow rows.

Weekly Average	Single Day
Rank - Source IP : Flows Count	Rank - Source IP : Flows Count
"1 - 161.223.1.106 : 600315.5"	"1 - 161.223.1.106 : 631251"
"2 - 159.152.145.176 : 380985"	"2 - 159.152.145.176 : 470112"
"3 - 161.223.1.142 : 366744"	"3 - 161.223.1.142 : 388523"
"4 - 161.223.1.108 : 341350.25"	"4 - 161.223.1.108 : 353272"
"5 - 190.49.180.97 : 242221"	"5 - 190.49.180.97 : 272309"
"6 - 161.223.1.164 : 210310.25"	"7 - 161.223.1.164 : 219430"
"7 - 161.222.192.123 : 168434.75"	"8 - 161.222.192.123 : 170911"
"8 - 191.220.233.80 : 153681.75"	"9 - 191.220.233.80 : 163650"
"9 - 190.49.138.164 : 122484.25"	"11 - 190.49.138.164 : 129751"
"10 - 64.75.33.200 : 87513"	"13 - 64.75.33.200 : 88752"
"11 - 162.185.32.105 : 71962"	"12 - 162.185.32.105 : 122229"
"12 - 65.126.59.119 : 65448.25"	"15 - 65.126.59.119 : 63652"
"13 - 190.49.135.29 : 64380.75"	"14 - 190.49.135.29 : 67519"
"14 - 191.220.195.154 : 61602.5"	"16 - 191.220.195.154 : 59857"
"15 - 192.239.62.2 : 61190.75"	"6 - 192.239.62.2 : 242587"
"16 - 159.152.49.236 : 54384.75"	"17 - 159.152.49.236 : 59236"
"17 - 162.185.32.85 : 48611"	"10 - 162.185.32.85 : 162072"
"18 - 190.118.118.242 : 48496.75"	"19 - 190.118.118.242 : 46193"
"19 - 190.118.162.234 : 45481.25"	"18 - 190.118.162.234 : 53868"
"20 - 195.168.62.200 : 43496.5"	
	"20 - 191.220.108.14 : 45501"

Figure 5.11: Filtering suspicious source IP addresses through Top rank

A summary of selected suspicious source and destination IP addresses are listed below:

```
Sa: ["159.152.145.176","162.185.32.105","192.239.62.2","162.185.32.85","191.220.108.14"]

Da: [192.239.62.2","190.49.180.97","161.222.192.123","162.185.32.105","190.49.135.29","190.119.161.4"]
```

Significant Terms Aggregation

Top ranks can filter out IP addresses with large number of flows. But those IP addresses which may have significant peak increase during the suspicious time slot but without huge flows will be missed. The new idea is that it is very helpful to find some IP addresses which only have frequent appearance during suspicious time but steady little appearance at other days.

Through researching, Elasticsearch has an advanced aggregation - "Significant Terms Aggregation"[19] can achieve this goal. This aggregation can pick the terms that have undergone a significant change in popularity measured between a foreground and background set[19]. In this case, all the flows of "10:00-20:00" on 18th are foreground data set while flows of "10:00-20:00" on 17th are used as background data set. The Elasticsearch request is shown in Source Code 5.2.

Source Code 5.2 Significant terms aggregation for filtering suspicious source IP addresses

```
curl -XGET localhost:9200/logstash-20120118, logstash-20120117/netflow/_search?pretty  -d '
{
    "query": {
        "bool": {
             "filter": [
                 { "terms" : {"pr" : ["TCP"]} },
                 { "range" : { "ts": {"gte": "2012-01-18 10:00:00.000", "lte": "2012-01-18 20:00:00.000"}}}
             ]
        }
    },
    "aggs" : {
        "SIGNIFICANT_SA": {
            "significant_terms" : {
                "field" : "dda",
                "min_doc_count":10000,
                "exclude" : [],
                "size" : 20,
                "background_filter": {
                    "terms" : {"pr" : ["TCP"]}
                },
                "background_filter": {
                    "range" : { "ts": {"gte": "2012-01-17 10:00:00.000", "lte": "2012-01-17 20:00:00.000"}}
                }
            }
        }
    }
}'
```

"min_ doc_count" parameter allows to only select the IP addresses with a specified minimum flows number. "size" parameter limit to only select top rank N IP addresses among them. These rules can help only select most suspicious IP addresses with big changes in the frequency of occurrence. "exclude" can used to exclude some IP addresses from the results.

Since still IP addresses with larger increase will have bigger influence and may reflect more features. The results of suspicious source IP addresses with minimum flows of 50000 and 10000 are generated as below:

```
Total flows count >= 50000:
Sa: ["192.239.62.2","162.185.32.85"]
Da: ["192.239.62.2","161.222.192.123","190.49.180.97"]

Total flows count >= 10000:
Sa: ["213.123.113.144","191.220.137.159","161.223.65.19","91.171.235.167"]
Da: ["192.238.9.157"]
```

5.3.2 Verifying Suspicious Flows

To be able to know the real reasons for that big increase, it is necessary to look into the details of those suspicious IP addresses selected through the former step. But it is found there exists some false positive suspicious IP addresses, which will affect the surveying. So the survey will be carried with two steps:

Eliminating False Positive IP addresses
The true positive IP addresses should show an obvious peak behavior at the suspicious time slot on the 18th. After many different attempts, It is found that the "split vertical bar chart" can help select true positive IP addresses conveniently. This chart is essentially the visualization of the date histogram aggregation with a nested terms aggregation. The Figure 5.12 and Figure 5.13 are the diagrams for those selected source IP addresses.

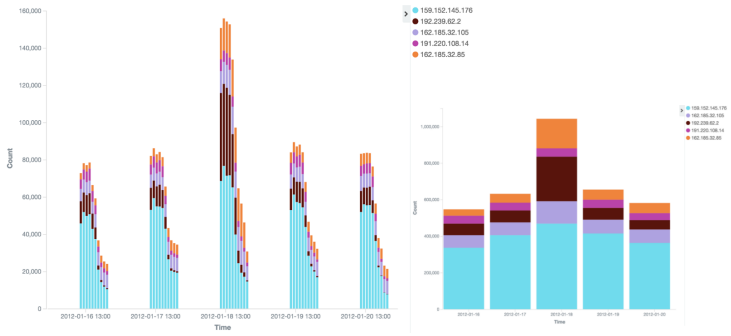


Figure 5.12: Split Vertical Bar Chart of suspicious source IP addresses - 1

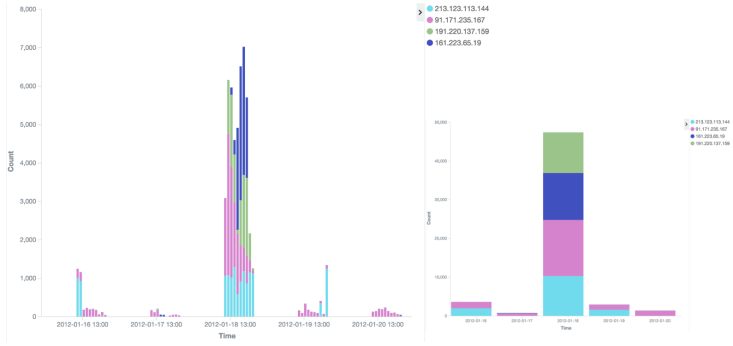


Figure 5.13: Split Vertical Bar Chart suspicious source IP addresses - 2

IP addresses with similar magnitude of flows are drawn in the same diagram. As seen from them, some of those IP addresses have obvious abnormal increase on 18th while some of them have just slightly increase which should be considered acceptable. So the positive IP addresses are summarized as:

```
sa ["192.239.62.2","162.185.32.85"]
   ["213.123.113.144","191.220.137.159","161.223.65.19","91.171.235.167"]

Da ["192.239.62.2","161.222.192.123","190.49.180.97","190.119.161.4"]
   ["192.238.9.157"]
```

Surveying Flows Details

Now, it is time to look into the flows of those true suspicious IP addresses. Surveying flows details can just use the "Daily Dashboard" built before by adding a term filter to just watch all the flows of a single interested source or destination address. Through this process, two kinds of traffic behaviors are found:

– Most IP addresses have increased flows with large bytes as shown in Figure 5.14.

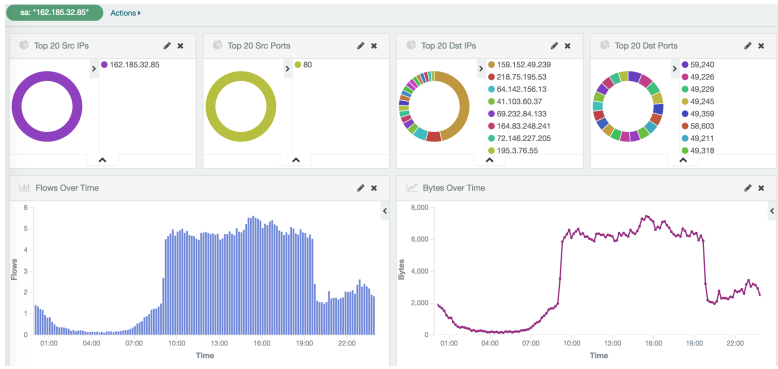


Figure 5.14: Suspicious TCP traffic behaviour -1

– For the destination address "161.222.192.123", there are increased flows and bytes with source ports "21" which is the File Transfer Protocol (FTP) control port for transferring data. As shown in Figure 5.15.

They all indicate data transfer. After confirming with UNINETT, it could be a large scale software update on 18th, e.g some new operating system or a new version of software, which fits the features than no sudden peak behaviors in the daily traffic but the whole network transferred larger amount of data than other normal days.

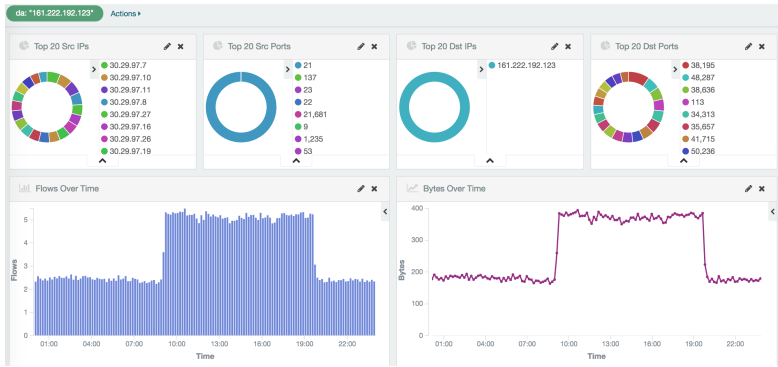


Figure 5.15: Suspicious TCP traffic behaviour -2

5.4 Use Case 3: Detecting Common Attacks

Common attacks are frequently happened. It is not easy to detect them directly from monitoring traffic diagrams. Also it is necessary to detect attacks positively rather than finding suspicious behaviors and trying to know which kind of attacks are they. In this thesis, it is also researched how ELK can help detect common attacks.

5.4.1 Spam Emails

NetFlow data can be used to detect spam email IP addresses. During researching, many thesises are found to design flow based spam detection methods. In this thesis [48], an algorithm based on the Simple Mail Transfer Protocol (SMTP) flows characters of spam email IP addresses is designed and tested, which turns out to be workable. The main idea is that several useful criteria based on the common behaviors of spam email IP addresses can be made for filtering malicious IP addresses. Inspired by this, the procedures of how ELK can detect spam email IP addresses are explored.

Surveying SMTP Traffic Behaviors
The first stage should be studying the SMTP traffic behaviors of the data set used for this thesis to summarize criteria. SMTP traffic patterns on 16th is studied on the top 100 source IP addresses with most distinct destination IP addresses to the port 25. Six kinds of parameters for a single IP are generated through Elasticsearch queries and aggregations:

- Outgoing SMTP connection count(a) and distinct destination IP addresses(b).
- Incoming SMTP connection count(c) and distinct source IP addresses(d).
- Incoming TCP connection count(e) and distinct source IP addresses(f).

Partial results are shown in Figure 5.16. Different colors stand for different kinds of

		a	b	c	d	e	f
1	190.119.203.185	1378,	462,	0,	0,	1076,	423
2	190.119.203.179	1358,	259,	1,	1,	1068	222
3	190.119.203.186	1345,	449,	0,	0	1015	386
4	190.118.200.101	1311,	291,	0,	0,	785	113
5	161.223.200.248	1252,	28,	0,	0,	397	8
6	190.119.109.77	1153,	361,	0,	0,	775	267
7	190.119.203.182	1129,	250,	1,	1,	918	247
8	190.119.109.114	991,	324,	0,	0,	690	254
9	65.233.160.172	835,	78,	0,	0,	337,	34
10	161.223.200.249	757,	25,	0,	0,	35	6
11	190.49.62.148	693,	339,	1466,	1058,	1797	1152
12	161.223.46.23	595,	2,	255,	1,	1149	34
13	213.5.56.153	592,	7,	0,	0	287,	6
14	190.49.35.107	548,	282,	0,	0	298	113
15	190.119.203.211	531,	147,	0,	0	375	113
16	170.75.13.68	530,	29,	222,	35	630	55
17	158.27.94.249	517,	173,	733,	467	1032	542
18	191.220.108.102	496,	242,	1,	1	376	139
19	190.119.203.210	493,	124,	0,	0	382	110
20	158.27.94.244	469,	161,	815,	570	417	149
57	76.91.51.214	108	33",	2,	2	59,	24
85	93.175.223.3	75	75	0	0	6	6

Figure 5.16: SMTP traffic behaviors

SMTP traffic behaviors:

- Firstly, spam IP addresses probably shouldn't have many incoming TCP connections with the source port 25. So those green rows should be normal email IP addresses. The other IP addresses all have zero or less than 5 incoming SMTP connections.
- But through drawing the line diagram of one week for outgoing SMTP traffic of them, many IP addresses which are marked as brown rows show regular traffic patterns as seen from the Figure 5.17. They probably are some IP addresses used for sending log messages to some servers or it is also possible that the port 25 is configured for some special usages. Besides, it is easy to notice in fact this kind of IP addresses are concentrated within several network subsets. So they probably are legal IP addresses.
- After filtering them out, the blue row with only 7 distinct destination IP addresses, also turns outs not spam IP addresses. The spam email IP addresses should have a larger number of distinct destination IP addresses.

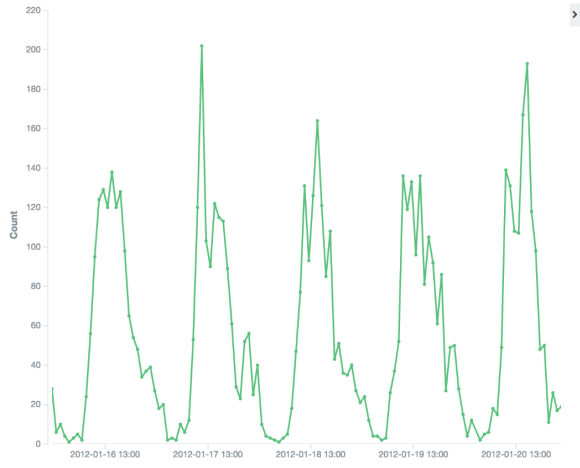


Figure 5.17: Regular SMTP traffic

– The purple row stands for a set of IP addresses with close number of outgoing SMTP connections and distinct destination IP addresses. It is found they have very few incoming TCP connections with source port 25. And for each destination IP addresses, there exists only one flow. They are probably port scans. As SMTP is based on TCP protocol. TCP communication needs three handshakes to establish a connection. So it should still have some incoming TCP connections with source port 25.

– Until now, only the red rows are left. The weekly traffic of "65.233.160.172" and "76.91.51.214" is shown in Figure 5.18 and Figure 5.19 with very suspicious peak behaviors. They are probably the spam email IP addresses.

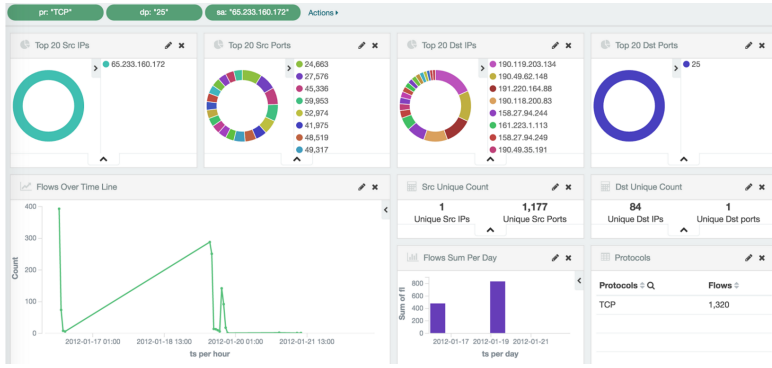


Figure 5.18: Suspicious Spam traffic - 1

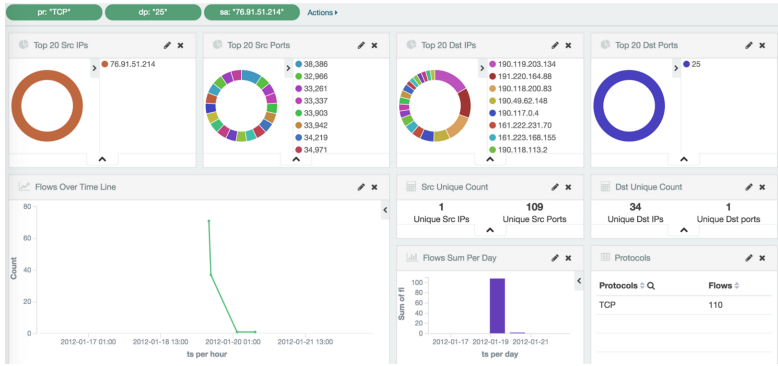


Figure 5.19: Suspicious Spam traffic - 2

ELK spam detection

Through the surveying steps, it shows for spam IP addresses, there are several criteria probably should be matched:

1. Distinct destination IP addresses$>$ 20
2. Incoming SMTP connections $<$ 3
3. Not port scan
4. Not known legal IP addresses
5. Irregular traffic patterns

The numerical value 20 and 3 are decided based on the experimental data set and should be adjusted accordingly. To realize the complete detection, the needed Elasticsearch requests are listed below:

– Select suspicious spam IP addresses. The initial search used before only selects top 100 IP addresses with most outgoing SMTP connections. Now some more conditions can be added to this initial search. Criteria 4 can be realized using range filters to not select the IP address within the possible legal subnets of those brown rows before, as seen in Source Code 5.3.

Source Code 5.3 Select suspicious spam IP addresses - Search part

```
"query": {
  "bool": {
    "must_not": { "range" : { "sa" : { "gte" : "161.223.0.0", "lte" : "161.223.255.255"} }},
    "must_not": { "range" : { "sa" : { "gte" : "190.119.0.0", "lte" : "190.119.255.255"} }},
    "must_not": { "range" : { "sa" : { "gte" : "161.220.0.0", "lte" : "161.220.255.255"} }}
    "filter": [
      { "term" : { "dp": "25" }}
    ]
  }
}
```

Further, through "Bucket Selector Aggregation" [8] of Elasticsearch, criteria 1, 3 can be added to this selection as well. It is a pipeline aggregation which can execute a customer script which determines whether the current bucket will be retained in the results. "Distinct_IP > 20&& CONN_COUNT / Distinct_IP > 2" selections criteria is made. "Dictinct_IP" and "CONN_COUNT" are the results of former metric aggregations. The radio is for filtering "port scan" IP addesses. As SMTP is built on TCP, because of the three handshake of TCP protocol, the number of outgoing SMTP connections at least should be bigger than distinct IP addresses count, very probably should be bigger than the double size. The complete aggregation is in Source Code 5.4:

Source Code 5.4 Select suspicious spam IP addresses - Aggregation part

```
"aggs" : {
  "SMTP_IP" : {
    "terms" : {
      "field" : "sa",
      "size" : "0"
    },
    "aggs" : {
      "Distinct_IP" : {
        "cardinality": { "field": "da" }
      },
      "CONN_COUNT" : {
        "value_count" : { "field" : "sa" }
      },
      "SELECTOR": {
        "bucket_selector": {
          "buckets_path": {
            "Distinct_IP": "Distinct_IP",
            "CONN_COUNT": "CONN_COUNT"
          },
          "script": "Distinct_IP > 20 && CONN_COUNT / Distinct_IP > 2"
        }
      }
    }
  }
}
```

– Through the former selective searching, a smaller set of suspicious IP addresses are selected. The next step is to count the incoming SMTP connections to fit criteria 2 for distinguishing from legal SMTP IP addresses. So add those IP addresses as destination IP addresses filter and for each destination IP address the number of incoming connections is generated, as depicted in Source Code 5.5. "Bucket Selector Aggregation" is used as well.

– Finally the vertical bar charts used before can realize the criteria 5.

So detecting suspicious spam email IP addresses can be realized in three steps easily and quite fast through ELK. Also this approach is very flexible to change the current criteria and add new criteria with no extra pains.

Source Code 5.5 Select suspicious spam IP addresses-3

```
curl -XGET localhost:9200/logstash-20120119/netflow/_search?pretty  -d '
{
  "query": {
    "bool": {
      "filter": [
        { "term" : { "dp": "25" }},
        { "terms" : { "da":  ["65.233.160.172","170.75.13.68"......] }}
      ]
    }
  },
  "aggs" : {
    "SMTP_IP" : {
      "terms" : {
        "field" : "da",
        "size" : "0"
      },
      "aggs" : {
        "CONNECTIONS" : {
          "value_count": { "field": "sa" }
         },
        "SELECTOR": {
          "bucket_selector": {
            "buckets_path": { "CONNECTIONS": "CONNECTIONS" },
            "script": "CONNECTIONS < 3"
          }
        }
      }
    }
  }
}'
```

5.4.2 Port Scan

Port scan is a popular strategy used by attackers. For simplicity, only TCP port scan is researched. There are three types, as shown in Figure 5.20. The TCP vertical and horizontal scan seem to have obvious features to realize in Elasticsearch through the similar methods used in the spam detection. They are explored separately. As the destination ports "80", "443" are common websites ports normally with large amounts of visiting traffic, they should be filtered out.

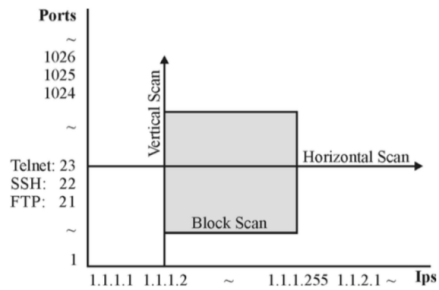


Figure 5.20: Port scan types [42]

For TCP vertical scan, the top 1000 source IP addresses with most destination IP addresses and top 1000 destination IP addresses with most destination ports are all studied aiming to find some behavior patterns. But no promising patterns were found and no suspicious IP addresses were found successfully. Among them, many IP addresses with large ports visiting turned out to be data transfer. The reason could be nowadays, port scan uses more advanced attacking techniques which help hide attack behaviors into normal traffic. So a vertical scan may be carried out by many source IP addresses and scanned very slowly. To detect them, more advanced specialized algorithms and systems are needed. Those complex calculations and statistics are out of the capabilities of ELK.

For TCP horizontal scan, the same situation is met. But horizontal scan usually targets a few of ports which probably are some common known ports. During the researching process, it is found several destination ports are visited most frequently which also turn out to be some vulnerable ports with some famous attacks appearing frequently, like ports 3389, 21, 22, 23, 1433, 4899, 445, 25. Detecting attacks of them could be done by directly watching the traffics of them.

Through researching, in Kibana, the "bubble chart" is very fitful for this purpose. This chart is essentially the visualization of the date histogram aggregation with the nested terms aggregation on destination ports. Compared with the line chart used before, it uses separate points to draw diagram and also different size of points stand for different number of results. Figure 5.21 and Figure 5.22 are the diagrams generated per day and per week. It is very easy to notice some bubbles with abnormal appearances, which stand for suspicious network situations on the related ports.

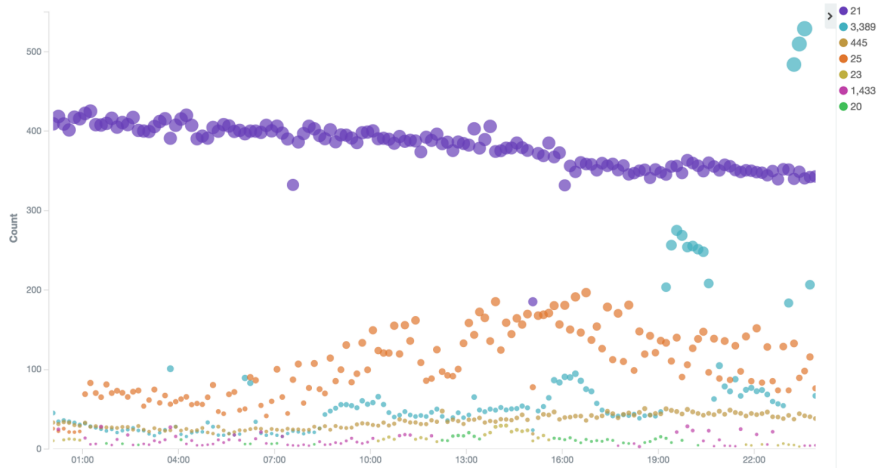


Figure 5.21: Bubble Charts -1

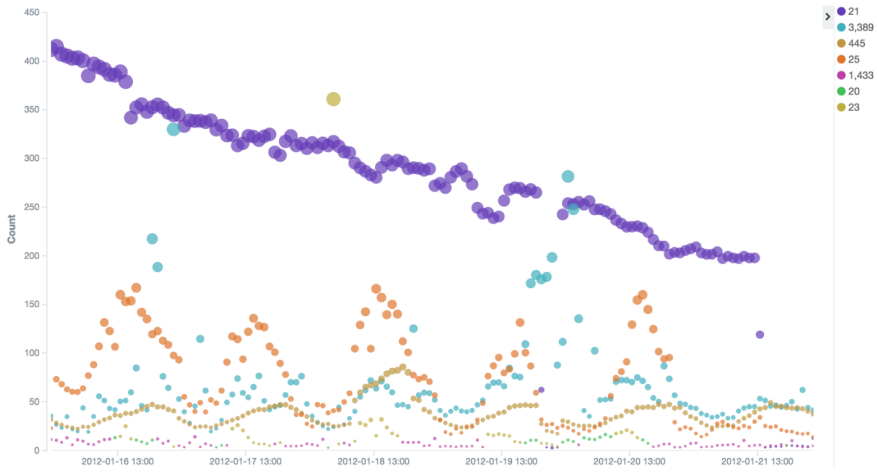


Figure 5.22: Bubble Charts -2

Chapter 6
Discussion

In Chapter 5, practical use cases of how ELK can manage large scale NetFlow data are presented. Based on the findings of the experiments process, two suggestions are discussed in this chapter which aims to help take the better use of ELK for the analyzing NetFlow data.

6.1 Providing Data for Other Systems

Through the three kinds of experiments before, it is easy to find ELK is better at narrowing down data and distinguishing suspicious flows according to some obvious features quickly. But not good at detecting complex attacks and usually the results are not determined. It is because the network anomaly techniques are becoming more and more sophisticated and even many advanced detection methods nowadays can't find them easily. In fact, there are many good specialized detection systems available. Maybe ELK can be used as a pipeline between a large amount raw NetFlow Data and advanced anomaly detection systems with the roles of conveniently storing and providing data. Two kinds of useful data could be provided:

First kind: to achieve faster and efficient data analysis, ELK could provide data selectively according to the specifics analysis purposes and requirements of users. This kind of data has been shown a lot in the Chapter 5, some examples are summarized below:

1. Data satisfying custom defined searching conditions.
2. Top rank statistics.
3. Data with suspicious peak behaviors.
4. Data satisfying custom criteria through script aggregations.

Second kind: except the various aggregations have mentioned so far, Elasticsearch also has some more advanced mathematics aggregations. Some mathematics aggregations have been used alongside within the former use cases for getting some useful numerical values, like the "Cardinality Aggregation" used before for getting the unique counts. They can generally be grouped into three categories:

1. Percentiles Aggregation [18] and Percentiles Rank Aggregation [17]
   The former one can be used to calculates one or more percentiles over numeric values while the second one show the percentage of observed values which are below certain value. For example the 60th percentile is the value which is greater than 60% of the observed values and if a value is greater than or equal to 60% of the observed values, it is said to be at the 60th percentile rank. When deciding some thresholds, the Percentiles Aggregation can be helpful to get the distribution of a set of numerical value.
   For example, when deciding the criteria number of the unique destination IP addresses and outgoing connections for suspicious spam IP addresses, the following aggregation in Source Code 6.1 is used.

Source Code 6.1 Percentiles aggregation example

```
"aggs" : {
  "TOP_IP": {
    "terms": {
      "field": "sa",
      "size": 0,
      "min_doc_count": 10
    },
    "aggs":{
      "UNIQUE_COUNT": {
        "cardinality": { "field": "da" }
      },
      "CONNECTIONS" : {
        "value_count" : { "field" : "da" }
      }
    }
  },
  "UNIQUE_COUNT_DISTRIBUTION": {
    "percentiles_bucket": {
      "buckets_path": "TOP_IP>CONNECTIONS",
        "percents": [20.0, 30.0, 40.0, 50.0, 60.0, 70.0]
    }
  },
  "CONNECTIONS_DISTRIBUTION": {
    "percentiles_bucket": {
      "buckets_path": "TOP_IP>UNIQUE_COUNT",
      "percents": [20.0, 30.0, 40.0, 50.0, 60.0, 70.0]"
    }
  }
}
```

The idea is that spam IP addresses should have relative larger those numbers than normal IP addresses, e.g. lager than 50%. Through "Terms Aggregation", for each source IP addresses with minimum 10 flows per day(primarily filtering out the IP addresses of personal users for sending emails), the unique

destination IP addresses and outgoing connections are calculated. Next, "UNIQUE_COUNT_DISTRIBUTION" and "CONNECTIONS_DISTRIBUTION" parts use "Percentiles Aggregation" to get the distributions of the former calculated unique destination IP addresses and outgoing connections counts.

2. Avg, Min, Max, Sum and Stats Aggregations [20]
   They are useful metric aggregations to calculate average, minimum, maximum and sum value of numeric values. Stats aggregation integrates the former aggregations and provide them together. They all can be used nested flexibly.
3. Derivative Aggregation [14] and Extended Stats Aggregations [15]
   They are pipeline aggregations which calculates more complex statistics like derivative, sum of squares, standard deviation. These statistics are helpful for monitoring the trend, speed of data changes etc.

6.2 Processing Elasticsearch JSON Results

The results of Elasticsearch requests through the "curl" command are in the JSON format. Although JSON format data is easy to understand, the interested information is often dispersed in the results. For example the aggregation result of "TOP 10 source IP addresses which talk to most number of distinct destination ports" is like:


```
"aggregations" : {
    "TOP_IP" : {
      "doc_count_error_upper_bound" : -1,
      "sum_other_doc_count" : 12686,
      "buckets" : [ {
        "key" : 3190928481,
        "key_as_string" : "190.49.180.97",
        "doc_count" : 545,
        "DISTINCT_COUNT_DP" : {
          "value" : 241
        }
      }, {
        "key" : 2715746702,
        "key_as_string" : "161.223.1.142",
        "doc_count" : 212,
        "DISTINCT_COUNT_DP" : {
          "value" : 204
        }
      }, {
        "key" : 2715746724,
        "key_as_string" : "161.223.1.164",
        "doc_count" : 97,
        "DISTINCT_COUNT_DP" : {
          "value" : 96
        }
      }
      ...
}
```

Only the IP addresses and the number of their distinct destination ports are the desired information. And during the experiments work of the thesis, it is found it is often needed to use those IP addresses as the filter conditions for other requests. Manual copy and paste them are not convenient.

To easy the task, some JavaScript code is made and tested to process the JSON results. The result before can be simplified easily as below:

```
["190.49.180.97","161.223.1.142","161.223.1.164","159.152.145.176","162.185.32.105",...]
[241,204,96,86,78,70,69,66,64,62]
```

In fact, Elasticsearch provides programming interfaces for several languages. Instead of using "curl" command to send requests to Elasticsearch, users can use programs to talk to Elasticsearch. One big potential benefit could be some automatic analysis programs could be made in this way.

Chapter 7
Summary and Further work

7.1 Summary

As NetFlow data plays crucial roles in network analysis, to solve the new requirements for large scale NetFlow data management in UNINETT, distributed NoSQL databases seem the right choice. This thesis project starts with surveying popular NoSQL databases and through evaluating various features of them based on the compliance to the requirements defined in Section 3.1, hopeful candidate systems are selected. ELK, which stands for the combination of Elasticsearch, Logstash and Kibana, stands out to be very promising for dealing with big NetFlow data.

To evaluate the performance, mainly three novel use cases of ELK for NetFlow data analysis are designed and practically experimented:

- Mornitoring Traffic Statistics in Section 5.2.
- Surveying Suspicious Flows in Section 5.3.
- Detecting Common Attacks in Section 5.4.

Based on the findings of the practical experimental work presented through the three novel use cases, the performance of ELK for large scale NetFlow data management is discussed below:

- With the integration of Logstash, importing NetFlow data into Elasticsearch is very convenient. The Logstash configuration file is very flexible for collecting, organizing data and furthers storing data into Elasticsearch.
- The way talking to Elasticsearch through a Restful API over HTTP is very clear. And the JSON encoded request body allows to build advanced queries with complex structures easily.

- Elasticsearch provides many powerful searches and aggregations. What's more, they can be grouped and nested together flexibly to provide even more powerful analysis. Searches and not very complex aggregations in Elasticsearch are executed very fast. Through tuning memory settings, the execution speed of very complex aggregations on large amount of data can also be fast enough.
- The combination of Elasticsearch and Kibana provides very good visualization based approaches for network traffic monitoring and quick troubleshooting.

Some suggestions of using ELK for NetFlow data analysis:

- ELK is better at monitoring the traffic patterns based on a relatively larger data set, e.g per day or per week to find abnormal traffic behaviors and selecting flows based on clear purposes. But it may not perform well on analyzing some complicated network behaviors accurately, for example detecting complex attacks.
- During the experimental work, it is found that the direct JSON results contains many useless words. So some simple Javascript code snippets are used to extract interesting information from them and display it in a clean and clear way. In fact, Elasticsearch supports to use several programming languages to access it. So maybe some programs can be made to help use Elasticsearch.

7.2 Further Work

- Due to limited time and for simplicity, all the experiments are carried out on a single server with limited data set. The distributed data management performance on very large data set needs to be further evaluated.
- The thesis project only works on historical NetFlow data. With Logstash, real time NetFlow data could be imported into Elasticsearch directly without operating on files. The real time NetFlow importing and analysis worth testing.
- Since MongoDB has native MapReduce function and MapReduce is good at carrying out advanced analysis, it will be useful to test it.

References

[1] "Introduction to Cisco IOS NetFlow-A Technical Overview". White paper, Cisco Systems, Inc. [Online; Last update: May 2012].

[2] "NetFlow Version 9 Flow-Record Format". White paper, Cisco Systems, Inc. [Online; Last update: May 2011].

[3] UNINETT AS. "About UNINETT AS". https://www.uninett.no/en/about-uninett. [Online; Accessed: Jun 19 2016].

[4] UNINETT AS. "Map of the UNINETT network". https://www.uninett.no/en/forskningsnettet/kart. [Online; Accessed: Jun 6 2016].

[5] Inc Cisco Systems. "Cisco IOS NetFlow". http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html. [Online; Accessed: Jun 6 2016].

[6] Manda Sai Divya and Shiv Kumar Goyal Goyal. "ElasticSearch: An advanced and quick search technique to handle voluminous data". *Compusoft*, 2(6):171, Jun 2013.

[7] Elastic.co. "Elasticsearch Reference [2.3] » Aggregations". https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations.html. [Online; Accessed: 14 Jun 2016].

[8] Elastic.co. "Elasticsearch Reference [2.3] » Bucket Selector Aggregation". https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-pipeline-bucket-selector-aggregation.html. [Online; Accessed: 11 Jun 2016].

[9] Elastic.co. "Elasticsearch Reference [2.3] » Cardinality Aggregation". https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-cardinality-aggregation.html. [Online; Accessed: 11 Jun 2016].

[10] Elastic.co. "Elasticsearch Reference [2.3] » Compound queries". https://www.elastic.co/guide/en/elasticsearch/reference/current/compound-queries.html. [Online; Accessed: 14 Jun 2016].

[11] Elastic.co. “Elasticsearch Reference [2.3] » Configuration”. https://www.elastic.co/guide/en/elasticsearch/reference/current/setup-configuration.html. [Online; Accessed: 11 Jun 2016].

[12] Elastic.co. “Elasticsearch Reference [2.3] » Date Histogram Aggregation”. https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-datehistogram-aggregation.html. [Online; Accessed: 11 Jun 2016].

[13] Elastic.co. “Elasticsearch Reference [2.3] » Date math support in index names”. https://www.elastic.co/guide/en/elasticsearch/reference/current/date-math-index-names.html. [Online; Accessed: 11 Jun 2016].

[14] Elastic.co. “Elasticsearch Reference [2.3] » Derivative Aggregation”. https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-percentile-rank-aggregation.html. [Online; Accessed: 11 Jun 2016].

[15] Elastic.co. “Elasticsearch Reference [2.3] » Extended Stats Aggregation”. https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-extendedstats-aggregation.html. [Online; Accessed: 11 Jun 2016].

[16] Elastic.co. “Elasticsearch Reference [2.3] » Getting Started » Basic Concepts”. https://www.elastic.co/guide/en/elasticsearch/reference/current/_basic_concepts.html. [Online; Accessed: 14 Jun 2016].

[17] Elastic.co. “Elasticsearch Reference [2.3] » Percentile Ranks Aggregation”. https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-percentile-rank-aggregation.html. [Online; Accessed: 11 Jun 2016].

[18] Elastic.co. “Elasticsearch Reference [2.3] » Percentiles Aggregation”. https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-percentile-aggregation.html. [Online; Accessed: 11 Jun 2016].

[19] Elastic.co. “Elasticsearch Reference [2.3] » Significant Terms Aggregation”. https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-significantterms-aggregation.html. [Online; Accessed: 11 Jun 2016].

[20] Elastic.co. “Elasticsearch Reference [2.3] » Stats Aggregation”. https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-stats-aggregation.html. [Online; Accessed: 11 Jun 2016].

[21] Elastic.co. “Elasticsearch Reference [2.3] » Sum Aggregation”. https://www.elastic.co/guide/en/elasticsearch/reference/current/

search-aggregations-metrics-sum-aggregation.html. [Online; Accessed: 11 Jun 2016].

[22] Elastic.co. “Elasticsearch Reference [2.3] » Term level queries”. https://www.elastic.co/guide/en/elasticsearch/reference/current/term-level-queries.html. [Online; Accessed: 14 Jun 2016].

[23] Elastic.co. “Elasticsearch Reference [2.3] » Terms Aggregation”. https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-terms-aggregation.html. [Online; Accessed: 11 Jun 2016].

[24] Elastic.co. “Kibana User Guide [4.5] » Introduction”. https://www.elastic.co/guide/en/kibana/current/introduction.html. [Online; Accessed: 14 Jun 2016].

[25] Elastic.co. “Logstash Reference [2.3] » Filter plugins » grok”. https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html. [Online; Accessed: 11 Jun 2016].

[26] Elastic.co. “Logstash Reference [2.3] » Logstash Introduction”. https://www.elastic.co/guide/en/logstash/current/introduction.html. [Online; Accessed: 14 Jun 2016].

[27] Elastic.co. “Logstash Reference [2.3] » Setting Up an Advanced Logstash Pipeline”. https://www.elastic.co/guide/en/logstash/current/advanced-pipeline.html. [Online; Accessed: 14 Jun 2016].

[28] Martin Fowler. “AggregateOrientedDatabase”. http://martinfowler.com/bliki/AggregateOrientedDatabase.html, Jan 19 2012. [Online; Accessed: 2 Jun 2016].

[29] Clinton Gormley and Zachary Tong. “Elasticsearch: The Definitive Guide [2.x] » Add an Index”. https://www.elastic.co/guide/en/elasticsearch/guide/current/_add_an_index.html, 2015. [Online; Accessed: 14 Jun 2016].

[30] Clinton Gormley and Zachary Tong. “Elasticsearch: The Definitive Guide [2.x] » Add Failover”. https://www.elastic.co/guide/en/elasticsearch/guide/current/_add_failover.html, 2015. [Online; Accessed: 14 Jun 2016].

[31] Clinton Gormley and Zachary Tong. “Elasticsearch: The Definitive Guide [2.x] » An Empty Cluster”. https://www.elastic.co/guide/en/elasticsearch/guide/current/_an_empty_cluster.html, 2015. [Online; Accessed: 14 Jun 2016].

[32] Clinton Gormley and Zachary Tong. “Elasticsearch: The Definitive Guide [2.x] » Coping with Failure”. https://www.elastic.co/guide/en/elasticsearch/guide/current/_coping_with_failure.html, 2015. [Online; Accessed: 14 Jun 2016].

[33] Clinton Gormley and Zachary Tong. “Elasticsearch: The Definitive Guide [2.x] » High-Level Concepts”. https://www.elastic.co/guide/en/elasticsearch/guide/current/aggs-high-level.html, 2015. [Online; Accessed: 14 Jun 2016].

[34] Clinton Gormley and Zachary Tong. “Elasticsearch: The Definitive Guide [2.x] » Life Inside a Cluster”. https://www.elastic.co/guide/en/elasticsearch/guide/current/distributed-cluster.html, 2015. [Online; Accessed: 14 Jun 2016].

[35] Clinton Gormley and Zachary Tong. “Elasticsearch: The Definitive Guide [2.x] » Scale Horizontally”. https://www.elastic.co/guide/en/elasticsearch/guide/current/_scale_horizontally.html, 2015. [Online; Accessed: 14 Jun 2016].

[36] Clinton Gormley and Zachary Tong. “Elasticsearch: The Definitive Guide [2.x] » Talking to Elasticsearch”. https://www.elastic.co/guide/en/elasticsearch/guide/current/_talking_to_elasticsearch.html, 2015. [Online; Accessed: 14 Jun 2016].

[37] Venkat N. Gudivada, Dantam Rao, and Vijay V. Raghavan. “NoSQL Systems for Big Data Management”. *2014 IEEE 10th World Congress on Services*, pages 190–197, Jun 2014.

[38] Venkat N. Gudivada, Dantam Rao, and Vijay V. Raghavan. “NoSQL Systems for Big Data Management”. *2014 IEEE 10th World Congress on Services*, pages 190–197, Jun 2014.

[39] Peter Haag. “Watch your Flows with NfSen and NFDUMP”. http://meetings.ripe.net/ripe-50/presentations/ripe50-plenary-tue-nfsen-nfdump.pdf, May 03 2005. [Online; Accessed: Jun 5 2016].

[40] Rick Hofstede, Pavel Celeda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. “Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX”. *IEEE*, 16(4):2037–2064, 2014.

[41] Internet Engineering Task Force (IETF). “IP Flow Information Export (ipfix)”. http://ietf.org/wg/concluded/ipfix.html, Mar 16 2015. [Online; Accessed: Jun 6 2016].

[42] Joris Kinable. “Detection of network scan attacks using flow data.”. *9th Twente Student Conference on IT*, Jun 2008.

[43] GIRISH KUMAR. “EXPLORING THE DIFFERENT TYPES OF NOSQL DATABASES PART II”. http://www.3pillarglobal.com/insights/exploring-the-different-types-of-nosql-databases, May 2014. [Online; Accessed: 2 Jun 2016].

[44] mike. “ELASTICSEARCH KIBANA BETA BEHIND APACHE PROXY”. http://mmbash.de/blog/kibana-beta-behind-apache-proxy/. [Online; Accessed: 11 Jun 2016].

[45] NoSQL-Database.Org. “LIST OF NOSQL DATABASES”. http://nosql-database.org/. [Online; Accessed: Jun 14 2016].

[46] SOURCEFORGE.NET. “NFDUMP DOCUMENTATION”. http://nfdump.sourceforge.net/. [Online; Accessed: Jun 6 2016].

[47] SOURCEFORGE.NET. “NfSen - Netflow Sensor’. http://nfsen.sourceforge.net/. [Online; Accessed: Jun 6 2016].

[48] Gert Vliek. “Detecting spam machines, a netflow-data based approach”. http://essay.utwente.nl/58583/, 2009.

[49] Hugh J Watson. “Tutorial: Big data analytics: Concepts, technologies, and applications”. *Communications of the Association for Information Systems*, 34:Article 65, 2014.

[50] Wikipedia. “Apache Hadoop”. https://en.wikipedia.org/wiki/Apache_Hadoop. [Online; Accessed: 2 Jun 2016].

[51] Wikipedia. “Database index”. https://en.wikipedia.org/wiki/Database_index. [Online; Accessed: 14 Jun 2016].

[52] Wikipedia. “MapReduce”. https://en.wikipedia.org/wiki/MapReduce. [Online; Accessed: 2 Jun 2016].