



BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

IP ADDRESS ACTIVITY MONITORING

SLEDOVÁNÍ HISTORIE AKTIVITY IP ADRES

BACHELOR'S THESIS

BAKALÁŘSKÁ PRÁCE

AUTHOR

AUTOR PRÁCE

KATEŘINA PILÁTOVÁ

SUPERVISOR

VEDOUCÍ PRÁCE

Ing. VÁCLAV BARTOŠ

BRNO 2016

Abstract

The volume of generated network traffic continually grows. In order to query data inside the traffic, an effective system of indexing data is required. This thesis addresses this problem, specifically effectively storing data for a longer period of time and looking up this data representing activity of communicating IP addresses. The aim of this thesis is to design and implement a system that stores and visualizes IP address activity. Activity means whether given address generated traffic during a given interval or not. This information has a binary value and can be represented by one bit, which significantly reduces volume of queried data. The system consists of backend processing incoming flow records and storing address activity to binary storage. Furthermore, it contains a web server which reads stored activity and visualises it in the form of an image based on user's request. The user can specify an area they wish to examine in more detail in the interactive web interface.

Abstrakt

Poslední dobou se objem přenášených dat po síti neustále zvyšuje. K urychlení prohledávání dat je potřeba mít způsob jejich vhodné indexace. Tato bakalářská práce se zabývá tímto problémem, konkrétně ukládáním a vyhledáváním dat za účelem zjištění aktivity komunikujících IP adres. Cílem této práce je navrhnout a implementovat systém pro efektivní dlouhodobé ukládání a vizualizaci aktivity IP adres. Aktivitou je myšleno, zda daná adresa generovala provoz v daném intervalu či ne, tedy lze ji reprezentovat jediným bitem, což redukuje objem prohledávaných dat. Výsledný systém se skládá z backendu monitorujícího provoz a ukládajícího záznamy o aktivitě do uložiště a jejich parametry do konfiguračního souboru. Dále obsahuje webový server, který na základě požadavků uživatele data čte a vizualizuje ve formě obrázků. Uživatel může specifikovat oblast dat, kterou chce zkoumat podrobněji, pomocí interaktivního webového rozhraní.

Keywords

Network Monitoring, IP Address Activity, NEMEA, libtrap, IPFIX, Flow Analysis

Klíčová slova

monitorování sítě, aktivita IP adres, NEMEA, libtrap, IPFIX, analýza datových toků

Reference

PILÁTOVÁ, Kateřina. *IP Address Activity Monitoring*. Brno, 2016. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Bartoš Václav.

Rozšířený abstrakt

V poslední době je internet běžnou součástí života a objem přenášených dat je čím dál tím větší. Aby se urychlilo vyhledávání dat, je potřeba mít způsob, jak data vhodně indexovat. Jedním typem těchto dat jsou IP adresy komunikujících zařízení. Tato bakalářská práce se tímto problémem zabývá, konkrétně efektivním ukládáním a vyhledáváním dat za účelem zjištění aktivity komunikujících IP adres.

Cílem této bakalářské práce je navrhnout a implementovat systém pro dlouhodobé ukládání a vizualizaci aktivity IP adres. Aktivitou adresy je myšleno, pokud adresa v daném časovém intervalu generovala nějaký provoz či ne. Díky tomu, že má aktivita adres binární hodnotu, lze ji efektivně uložit do binárního uložště jako jediný bit. Takto se velikost uložště a objem dat prohledaný při vyhledávání efektivně sníží.

Výsledný systém obsahuje backend, který analyzuje příchozí toky a na konci každého intervalu ukládá záznamy o aktivitě adres patřících do zvoleného rozsahu do binárního uložště. Informace o parametrech uložště jsou uchovávány ve speciálním konfiguračním souboru. Dále obsahuje webový server, který pomocí konfiguračního souboru načte záznamy o aktivitě z uložště a na základě požadavků uživatele ve webovém rozhraní vizualizuje tyto záznamy a pošle je webovému klientovi.

Uživatel si poté pomocí interaktivního webového rozhraní dostane informace o záznamech a může zvolit konkrétní oblast ze zobrazované aktivity, kterou může zkoumat podrobněji.

V kapitole 2 jsou popsány principy monitorování a NEMEA framework, ve kterém je implementován backendový modul. Ve 3. kapitole je detailněji popsán design výsledného systému a použitých technologií pro jednotlivé části. 4. kapitola je věnována popisu implementace jednotlivých částí systému, detailnější vysvětlení nejdůležitějších jejich funkcí. Dále je zde popis struktury a určení míry efektivity binárního uložště. Kapitola 5 je věnována popisu užitých technik testování a jejich výsledků. V kapitole 6 je vyhodnocení výsledků této bakalářské práce a návrhy na možná další vylepšení tohoto projektu, která budou realizována v budoucnu.

IP Address Activity Monitoring

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Ing. Václav Bartoš. The supplementary information was provided by Ing. Tomáš Čejka. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....

Kateřina Pilátová

May 16, 2016

Acknowledgements

I wish to express my sincere thanks to my supervisor, Ing. Václav Bartoš, for the guidance and encouragement. I am also grateful to Ing. Tomáš Čejka, my consultant, for providing me with all the necessary facilities, sharing expertise and guidance.

© Kateřina Pilátová, 2016.

This thesis was created as a school work at the Brno University of Technology, Faculty of Information Technology. The thesis is protected by copyright law and its use without author's explicit consent is illegal, except for cases defined by law.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 3 |
| 2 | Network Monitoring | 5 |
| 2.1 | The Purpose of Monitoring | 6 |
| 2.1.1 | NfSen | 6 |
| 2.2 | Types of monitoring | 6 |
| 2.3 | Flow | 6 |
| 2.4 | Network Measurements Analysis (NEMEA) Framework | 7 |
| 2.4.1 | Architecture | 7 |
| 2.4.2 | UniRec | 8 |
| 2.4.3 | Libtrap | 9 |
| 2.5 | Related Work | 9 |
| 3 | Application Design | 10 |
| 3.1 | System Architecture | 10 |
| 3.2 | Input and Output Data | 11 |
| 3.2.1 | NEMEA module | 11 |
| 3.2.2 | Web server | 11 |
| 3.2.3 | Web Client | 11 |
| 3.3 | Result format | 13 |
| 3.4 | Used Technologies | 14 |
| 3.4.1 | NEMEA Module | 14 |
| 3.4.2 | Web Server | 15 |
| 3.4.3 | Web Client | 15 |
| 3.4.4 | Conversion of a bitmap to an image | 16 |
| 4 | Implementation | 17 |
| 4.1 | Backend | 17 |
| 4.1.1 | Workflow | 17 |
| 4.1.2 | IP Address Data Type | 24 |
| 4.2 | Bitmap Storage | 25 |
| 4.3 | Web Server | 28 |
| 4.3.1 | Workflow | 28 |
| 4.4 | Visualisation Handler | 31 |
| 4.4.1 | Loading Configuration | 31 |
| 4.4.2 | Functionality | 31 |
| 4.5 | HTTP Requests and the Corresponding Responses | 33 |
| 4.5.1 | Main Page | 34 |

| | | |
|------------------------------|--|-----------|
| 4.5.2 | Updating Image | 34 |
| 4.5.3 | Area Selection | 35 |
| 4.5.4 | Index Calculation | 35 |
| 4.6 | Web Client | 36 |
| 4.6.1 | Page Content | 36 |
| 4.6.2 | Client Interactivity | 42 |
| 5 | Testing | 45 |
| 5.1 | Manual Testing | 45 |
| 5.2 | Unit Tests | 46 |
| 5.3 | User Testing | 46 |
| 5.3.1 | The Backend and the Web Server | 46 |
| 5.3.2 | Web Client | 47 |
| 5.4 | Results | 48 |
| 5.4.1 | Performance | 48 |
| 5.4.2 | Storage Size Comparison | 48 |
| 5.4.3 | Lookup Time Comparison | 49 |
| 6 | Conclusion | 51 |
| | Bibliography | 53 |
| | Appendices | 55 |
| List of Appendices | | 56 |
| A | CD Content | 57 |
| B | Installation Guide | 58 |
| B.1 | Requirements | 58 |
| B.2 | Additional Packages | 58 |
| B.3 | Running the System | 59 |

Chapter 1

Introduction

Over the last decades, exchanging data over the Internet has become part of everyday life. As the usage of networking became greater, the design of the networks became harder and so did securing network communication. Network administrators monitor networks to maintain an overview of the network status, including used resources, users' activities or types of devices connected to the network. Gathered data then helps administrators and designers make the network more suitable for their needs and increase the quality of the user experience. One of the collected parameters are the IP addresses of the communicating endpoints, which is the main concern of this bachelor's thesis.

A big volume of traffic is generated every day. Traffic consists of flow records carrying data between endpoints. In order to make data lookup faster, a special way of indexing data is required. The thesis addresses this problem and its purpose is to make storing and querying activity of IP addresses more efficient.

The aim of this thesis is to design and implement a system for storing and visualising IP activity. The activity means whether traffic was generated by/sent to a given IP subnet in the monitored interval or not. The resulting system consists of a backend and a frontend. The backend is a NEMEA framework module that stores information about IP address activity over a longer period of time in binary storage and the data statistics to a dedicated configuration file. In order to reduce the lookup time, the activity of each subnet is represented by one bit. Transforming stored data into images and providing information for the frontend is the role of the web server. The frontend, a web client, visualises the final images and interacts with the user. The user can then examine his areas of interest in more detail by selecting an area in the whole image.

This thesis is divided as follows. In Chapter 2, network monitoring is discussed and the NEMEA framework is introduced as it is the used framework for the backend. In Chapter 3, attention is brought to the application design, which is described in detail and its aspects are evaluated. In Chapter 4, the components of the implemented module, backend, web server and graphical user interface, are defined and the essential sections and principles are explained. Additionally, the structure the binary storage and its effectiveness is thoroughly discussed. In Chapter 5, testing methods of the module on both online and offline data are demonstrated and the results are examined and any problems encountered and their

solutions are discussed. Chapter 6 is dedicated to a discussion of the results, the thesis summary and suggestions for possible improvements that could be added in the future.

Chapter 2

Network Monitoring

The purpose of this chapter is to give the reader a theoretical background to network monitoring and the framework used for implementing the module.

Network monitoring is a part of network management that is used for maintaining the network status. A network monitoring system keeps track of devices and services based on traffic carried over the network. An example of an engagement of such a system can be seen in Figure 2.1.

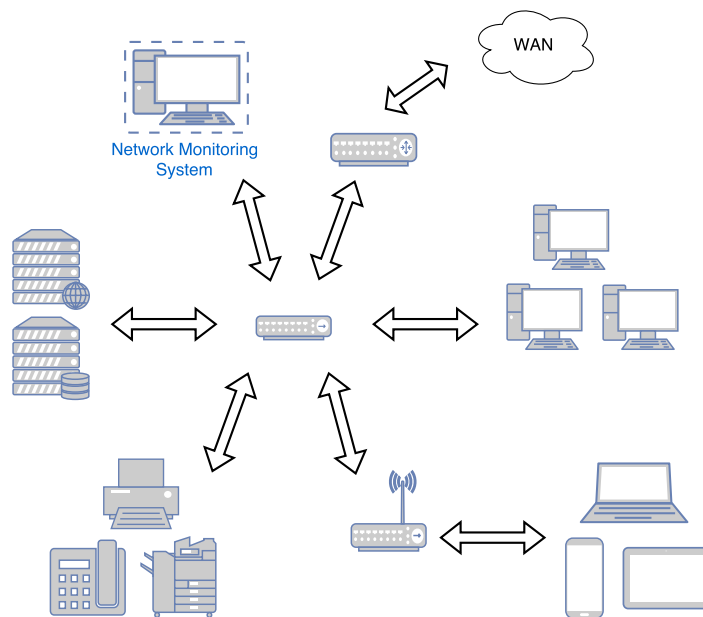


Figure 2.1: Example of monitored network containing Network Monitoring System

The target networks can be of different sizes (Local Area Network or Wide Area Network) and types (wired or wireless) and can consist of logically separated subnetworks.

2.1 The Purpose of Monitoring

Network monitoring has many uses in various areas. It can be used for detecting failing or malfunctioning components of the network that need to stay running on various devices, such as servers, routers, desktops or switches. It also helps with cost saving by finding redundant resources or identifying so-called *bottlenecks*. Moreover, it can be used to improve performance by measuring latency and spotting overloaded devices. Another example of monitoring usage is finding patterns that could signify security threats by analysing traffic.

2.1.1 NfSen

An example of a frontend for netflow [16] tools for storing and processing flow records called nfdump [8] is NfSen [9]. It is an interface that displays how much data was transmitted over the probe in a certain time period. However, in order to find out the most active IP addresses, the user has to display the whole stored time interval. To make the process faster and easier, the IP activity monitoring system can be used for displaying activity of the address space in the whole time interval and for showing which addresses from the address space were active at any the given moment. NfSen can then use the gained information to display how much traffic these addresses generated.

2.2 Types of monitoring

Monitoring can be either passive or active [22]. In active monitoring, the server continually polls for devices, applications and links status. If a device does not respond in time, it is considered unavailable. An example of a protocol used for active monitoring is the Internet Control Message Protocol (ICMP) [27]. Passive monitoring only gathers information about the network that is carried over the listening port of the monitoring probe. Information is sent in logs (for instance used in syslog [21]) or as data from network exporters (like in the case of the IP Flow Information Export (IPFIX) protocol [28]).

2.3 Flow

Nowadays, a big volume of data is being transferred through the network. In order to make the processing and transfer of captured traffic easier, monitoring systems are *flow-based* – probes firstly aggregate packets into *flows* [24].

A flow is a set of IP packets passing an *observation point* in the network during a certain interval of time [28]. Packets that belong to the same flow share some properties in L3 (Network Layer), L4 (Transport Layer) or in some cases even L7 (Application Layer) headers¹. For instance, in the case of NetFlow, these parameters are the ingress interface,

¹Since the IP address is the only attribute relevant to this thesis, the whole OSI model is not discussed in

source and destination IP address, source and destination port, IP protocol and Type of Service.

2.4 Network Measurements Analysis (NEMEA) Framework

In this bachelor's thesis, the implemented module can be used in a system called NEMEA.

NEMEA is a modular system that allows for the custom analysis of traffic gathered both on-line and off-line. Besides the Network and Transport Layer, of OSI Model analysis, it also enables the examination of Application Layer (L7) data.

NEMEA processes data *stream-wise* – data arrives as individual records, one by one, unless it is explicitly defined otherwise. Incoming records are analysed by the system without being stored aside along the way².

2.4.1 Architecture

An example of a monitoring system architecture using NEMEA can be seen in Figure 2.2:

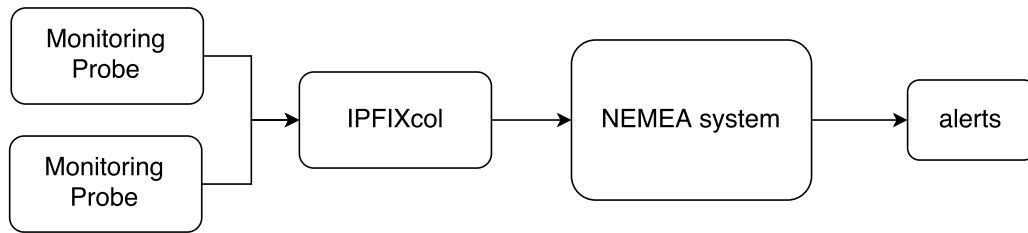


Figure 2.2: Architecture of Monitoring system including NEMEA

The system works as follows. The network is being monitored by monitoring probes (flow exporters). Probes aggregate packets using IPFIX [28] or NetFlow [16] technology and export flows to a central collector called IPFIXcol [17]. The collector converts received flows into another format called the Unified Record (UniRec) [17] (explained in section 2.4.2) and sends final flows to NEMEA.

NEMEA consists of modules that are assembled according to user's needs. Each module can have multiple ingress and egress one-way TRAP (further discussed in section 2.4.3) interfaces [30]. An example of a set of interconnected modules is in Figure 2.3 where *Module 1* has two egress interfaces that are connected to the only *Module 2* and *Module 3* ingress interface.

detail.

²The only case of storing incoming data is when it is the purpose of the particular participating module.

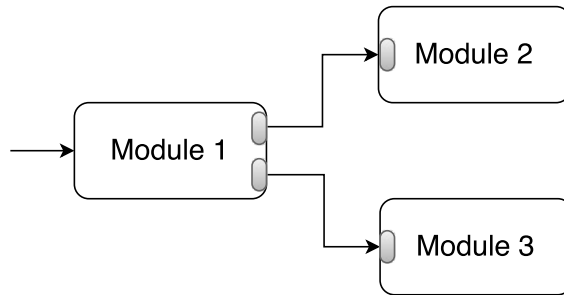


Figure 2.3: Interconnected modules

Modules can be implemented in C or C++ and a wrapper for Python is available. The aim of this bachelor's thesis is to implement one such module.

2.4.2 UniRec

The UniRec format supports flow extension by Application Layer data and as a result, makes accessing data in this layer easier. The structure of this format can be seen in Figure 2.4.

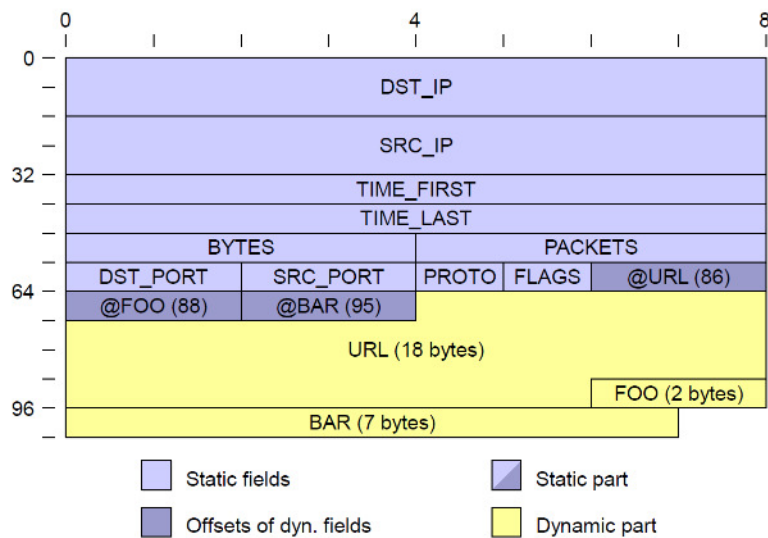


Figure 2.4: UniRec Format

UniRec defines offsets for accessing fields in various application layer protocols. Due to the diversity in the data sizes of individual protocols, this part of the structure changes dynamically to correspond to the used protocol.

2.4.3 Libtrap

TRAP is a library (also called *Libtrap* [19]) that implements common communication interfaces (IFC) used for communication among all NEMEA modules as seen in Figure 2.5. Communication is realised using either TCP sockets (TCP/IP IFC) or Unix sockets (UNIX IFC).

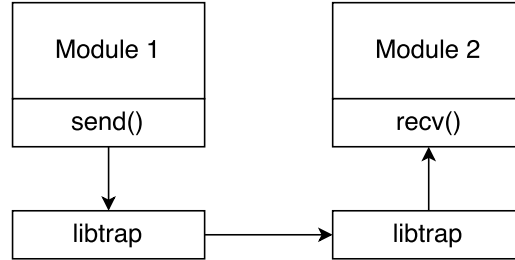


Figure 2.5: NEMEA inter-module communication

Libtrap contains features for controlling the communication and abstracts modules from the complexity of the underlying interfaces.

2.5 Related Work

The method of visualizing traffic in the form of an image is quite common. Quite a few projects are focused on providing graphic information about communication on the network. One of the similar projects is called *Interactive Visualisation for Network and Port Scan Detection* [23].

This project does not only examine IP activity but also includes port activity. This approach does include more information about the monitored flows and offers more ways of visualising the stored data but the size of the stored data is increased by the additional information which leads to a slower lookup time. This thesis is focused solely on one attribute that can be used in other networking tools (for example NfSen 2.1.1) as a key for looking up the rest of the data, including but not limited to the above-mentioned ports.

Chapter 3

Application Design

This chapter discusses the design of the resulting system for visualizing IP address activity. It introduces all system components and their functions, discusses technologies used for realizing the system and suggests two main modes in which the system can be run.

3.1 System Architecture

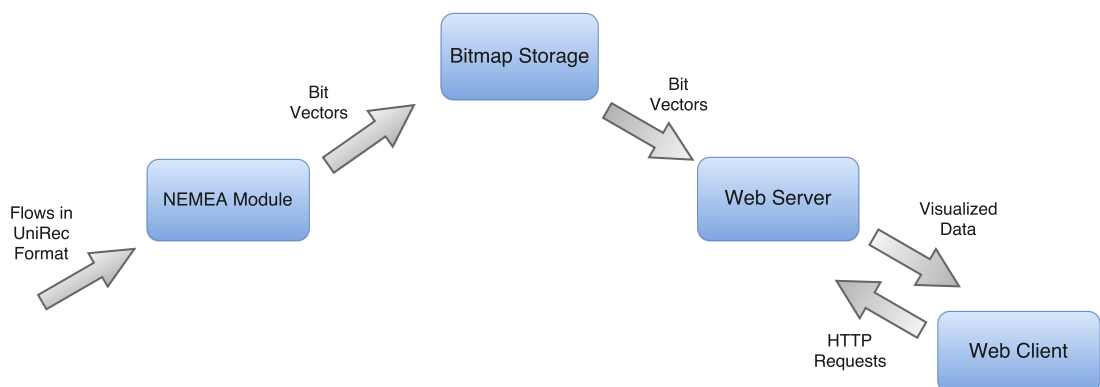


Figure 3.1: System Design

The system is illustrated in Figure 3.1. It consists of a NEMEA module, bitmap storage and a web client and server. The Figure 3.1 also shows the forms of data transmitted between individual components or method for.

The module processes received data and saves it to the bitmap storage, represented by a binary file. The web client sends a request for visualisation of IP Activity to the server. The web server receives the request, retrieves data from the storage based on inserted parameters and sends them back to the client. The web server runs independently of the

module, so that at any given moment, the configuration file must be up to date with the bitmap file state.

3.2 Input and Output Data

The main components ensuring the functionality of the system are the NEMEA module processing flows and the web server providing visualisation to a web client. In this section, their possible inputs and outputs illustrated in Figure 3.1 are discussed.

3.2.1 NEMEA module

For the NEMEA module, the input always consists of serialised flows that are received on its input interface in UniRec format. The input data can be retrieved either directly from online traffic flowing through the probe or offline data generated from a file. The output of the module is a bit vector representing activity of specified addresses (subnets) in a given period of time and it is written to a dedicated binary file, followed by updating the configuration file.

3.2.2 Web server

The web server listens on a predefined port and waits for requests from the client. The HTTP requests may vary but they typically consist of a URI containing several parameters that affect the server response. The response of the server is requested information in form of a response header value or an image (created from bitmap, see section 3.3) reflecting the IP activity and is sent back to the client.

3.2.3 Web Client

The client displays activity statistics and characteristics to the user. All the displayed data is received from the server based on client's requests.

The Expected Users

The group of users that are likely to interact with the interface is expected to be familiar with the principles of networking and the NEMEA framework. Therefore, this interface does not need any extended user guide, so the only instructions describing the usage can be found at the About section in the main menu.

The likely user scenarios are as follows:

- User seeks to discover which IP subnets were active at a particular time. The user provides the system with a file containing records from the desired time window.
- User wants to discover at which times the specified address space is/was active.
- User wants to examine an IP subnet or time interval activity more closely. They start the system multiple times with continually decreasing subnet/interval size to find out more detailed information.

By gaining the information about the IP activity, the user uses this information to examine the activity in other ways using tools such as NfSen (described in Section 2.1.1).

Layout

The frontend layout is intended to be simple and readable for the user so that they are not distracted or slowed down when interacting with the web interface. Used colour combination has enough contrast to be easily readable. Selected options are displayed differently for better orientation.

The image colouring is selected in such a way that the active areas stand out – active places are displayed as white and inactive places as black. Undefined areas have a gray colour. The undefined value is used when the currently monitored activity has a lower interval range than the time window or when displaying a selected area. By doing so, the image proportions stay the same (time window x number of subnets in the address space). The size of the image can be changed by selecting a ratio.

In order to avoid mixing up values from the background, the image is surrounded by a frame of a different colour.

Design of the layout structure is shown in Figure 3.2.

In the layout, the top part is dedicated to the original monitored area and visualises the whole bitmap. The bottom part is hidden until the user selects an area. All the Options are situated in the same section (displayed as green coloured). The user can change the type of the original bitmap and its scale at any time. In addition to this, they can also select an area by inserting minimum and maximum values of the individual axes.

Only after submitting the specified area, the bottom section is displayed. Like in the top section, the area characteristics and current position are available to the user, followed by the visualisation of the selected area.

The *Current Position* section offers the user the ability to see the values of the coordinates their cursor is pointing at. It is available for both the original image and the selected area. The background of the *Current Position* field is adapted to the colour of currently visited subnet activity at a particular moment. Both subnet and time interval are displayed below the field title.

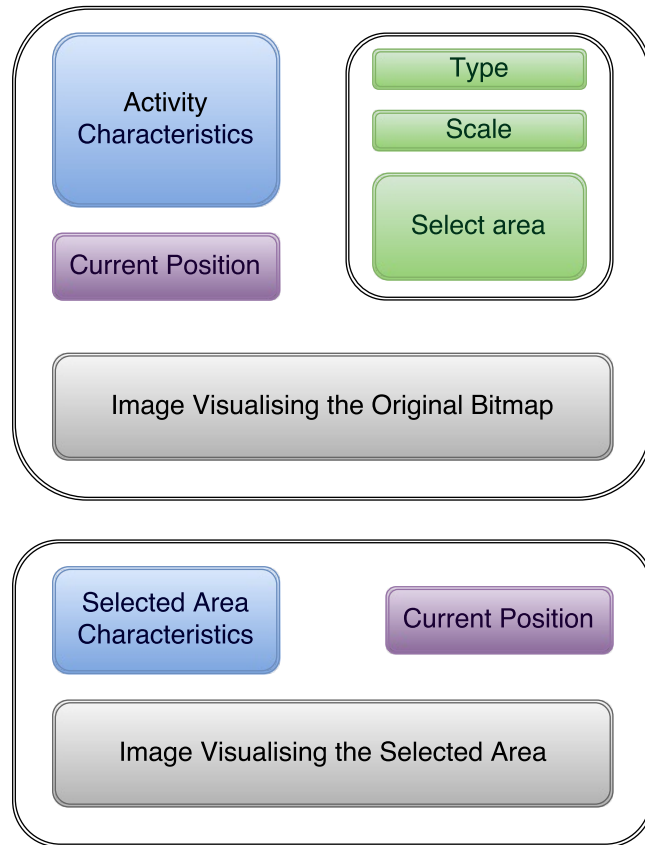


Figure 3.2: Frontend Layout

3.3 Result format

The result of the visualisation process is an image, or a set of images, that represent IP address activity of communicating endpoints. In order to cover all usage possibilities, separate images showing activity of monitored source IP addresses, destination IP addresses and monitored links (source and destination pairs) are provided to the user.

An example of such image can be seen in Figure 3.3. This image is a result of visualising activity at the nemea collector. For better readability, the image was captured in scaling 2:1. It contains an undefined area (gray coloured) as the time window is not filled with activity record yet.

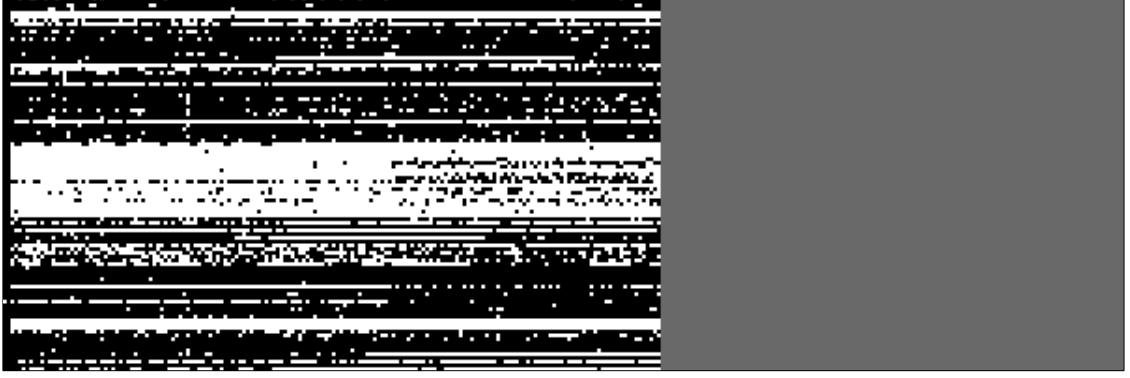


Figure 3.3: Visualised Activity

The x -axis represents time divided into defined units. The y -axis represents the range of predefined IP addresses to be monitored. The principle of displaying measured activity can be defined as follows:

$$\begin{aligned}
 &\text{for } i \in \langle addr_{min}, addr_{max} \rangle, j \in \langle t_{min}, t_{max} \rangle : \\
 &\quad x_{ij} = (1)_2 \Leftrightarrow \text{address } x_i \text{ is active in time interval } j \\
 &\quad x_{ij} = (0)_2 \Leftrightarrow \text{address } x_i \text{ is inactive in time interval } j
 \end{aligned} \tag{3.1}$$

3.4 Used Technologies

3.4.1 NEMEA Module

For implementing the NEMEA module, languages Python, C or C++ can be used.

When using Python, the level of abstraction is greater and so the programmer does not have an absolute control on how which abstract construction will be executed. As a result, the final module tends to be significantly slower due to unoptimised processing of a big volume of data in real time¹.

C++ offers more options and functionalities than C, including a special data structures for manipulation with bits (such as `std::vector<bool>` or `std::bitset<size>`). Furthermore, handling configuration file in YAML format is easier thanks to *YAML-cpp* [15] parser and emitter². Therefore, the C++ language is used for the implementation.

For communication with the module, *libtrap* [19] is used. Manipulating the received data is realised using *UniRec* format. These technologies are discussed in detail in section 2.4.

¹This statement is based on earlier experience. When comparing execution of the same functionality written in C and in Python, the module in Python was ten times slower.

²The version of the package must be 0.5.3 which requires C++11 or *Boost* [3] libraries installed.

3.4.2 Web Server

PHP is mostly used as a server-side programming language for its universal usability and provided functions, e.g. communication with an SQL database. However, in this architecture, the bitmap storage is represented by a single binary file.

In the case of an SQL database, additional operations for connecting, retrieving and processing data from the database in the form of tables are needed. If heterogenous data are used, that can be an advantage. But in this system, the only types of data needed are the information about the specified ranges can easily be maintained in a dedicated configuration file, and the binary data itself.

Therefore, C++ or Python, each of which have their own libraries (or packages) needed for a web server, can be used to realise needed functionality.

Due to a convenient package called *BaseHTTPServer* [18] (*http.server* in Python 3) that manages HTTP functionality, the resulting web server is implemented in Python. The implementation is compatible with both 2.x and 3.x versions. For convenient work with various data types, the following non-standard packages are used:

- *beautifulsoup4* [2] Used for parsing and modifying HTML files.
- *bitarray* [2] Used for handling bit vectors.
- *ipaddress* [6] Package for convenient work with IP addresses.
- *Pillow* [10] The fork of *PIL* (Python Imaging Library) for work with images.
- *pyyaml* [12] Parses YAML files (used for work with the configuration file).

3.4.3 Web Client

When implementing a web client, traditional web technologies are used:

- *HTML* – defines the content of the web page
- *CSS* – specifies the layout of the web page
- *JavaScript* – describes the behaviour of the webpage, is often used asynchronously (AJAX)

The web client will use these technologies to achieve a well arranged graphical interface that visualises data for the user and enables the user to specify an area that is to be cropped from the original image and displayed separately. Thanks to AJAX, all requests are handled asynchronously and no page reload is needed in order to retrieve requested data. In order to access and modify elements easily, JavaScript library *jQuery* [7] is used.

3.4.4 Conversion of a bitmap to an image

The web server uses a special module for visualising the bitmap (more in section 4.4). This module uses a package for processing images called Pillow [10] derived from *Python Imaging Library* [11].

Chapter 4

Implementation

4.1 Backend

Backend is implemented in a source file `ip_activity.cpp`. In this part of the thesis, operations carried on in this source file and associated data structures are discussed.

Backend receives parameters about the final image from the user, and saves parameters to the configuration file, so that the server can index the stored data. The program then scans incoming flow records and stores activity of defined address space in each interval into the binary storage.

The `ip_activity.cpp` module works with a template containing at least three fields of the UniRec format – SRC_IP (source IP address), DST_IP (destination IP address) and TIME_FIRST (first occurrence of the flow).

In order to work properly, the module has to have writing privileges to the current folder where the storage files and configuration file are to be created or rewritten.

4.1.1 Workflow

The workflow of the backend module is demonstrated in the Figure 4.1. It captures the main operations of the module that are discussed in more detail in this section. The *Module Initialisation* state is the start state of the workflow, when the module receives parameters from the user. The end states are distinguished by a perimeter line. The module can terminate either when an error occurs (during parameter validation or flow analysis) or successfully (after receiving the interrupt signal or the termination message).

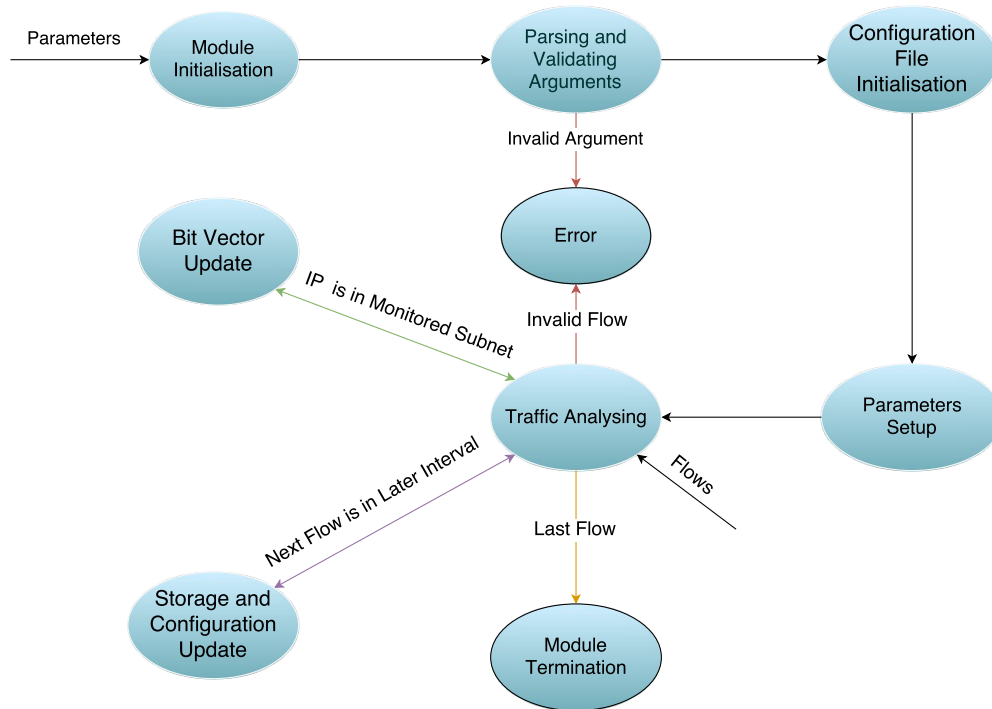


Figure 4.1: Backend Workflow

The module firstly receives arguments from the user, parses and validates them.

Parsing and Validating Arguments

Full list of possible parameters (excluding the libtrap common parameters) and their brief description:

- `-c/--config_file` Name of the configuration file. (default: "config")
- `-d/--directory` Directory for saving bitmaps and configuration. (default: ".")
- `-f/--filename` Name of bitmap files. (default: "bitmap")
- `-g/--granularity` Granularity of IPs (subnet size) by a netmask. (default: 8).
- `-r/--range` Range of monitored IPs in format <first ip>,<last ip>.
- `-t/--time_interval` Time unit for storing data in seconds. (default: 300)
- `-w/--time_window` Time window of stored data in intervals. (default: 100)

Other parameters that are common for all modules can be found when parameter `-h trap` is entered.

When the user runs the backend program, they can choose what parameters the original bitmap will have. The `--filename` specifies the name of binary storage files and is used as the node name in the configuration file in Section 4.1.1. The name of the configuration file can be defined by setting `--config_file`. If the configuration filename is changed, the changed name has to be passed to the web server that retrieves the storage statistics from it.

Both the bitmap files and the configuration file are stored to the same directory. By default, it is the current working directory but it can be changed by the user by setting `--directory` parameter. If a custom directory is set, it has to be at the server as well.

The user can also specify the address space, (`--range`) activity of which will be monitored and its granularity `--granularity`. If the address range is not entered, the whole address space of IPv4 is used. In order to create a reasonable amount of subnets, granularity default value is set to 8.

Both addresses of the address range must be of the same version and are transformed to the defined granularity during the work flow (example below in Listing 4.1). Thanks to this transformation, the subnets can be calculated using the netmask, the size of all subnets remains the same (whereas if the address space could have customised ending, the last subnet could end up smaller) and no additional customization has to be done.

```
./ip_activity -r "123.4.5.6,200.0.255.1" -g 16
```

```
Addresses passed to backend:          123.4.5.6 and 200.0.255.1
```

```
Addresses after transformation (/16): 123.4.0.0 and 200.0.0.0
```

Listing 4.1: Address Range Transformation

One time unit for defining subnet activity is specified by `--time_interval` in seconds. Time window `--time_window` specifies the maximum intervals stored into the past. After that, the oldest interval is rewritten by the first one as the storage acts like a circular buffer.

Example of usage:

```
./ip_activity -i t:12345 -r "1.2.3.4,4.5.6.7" -g 16 -w 250  
-f "my_bitmap" -t 100 -p
```

This sets up the module to store activity about IP addresses 1.2.3.4 to 4.5.6.7 by /16 subnets, has a time interval of 100 seconds and a time window of 250 intervals. The activity is stored in files with *my_bitmap* prefix.

Configuration File Initialisation

After that, based on the arguments, it opens a configuration file `<filename>.yaml` (if it does not exist, the module creates it first) and adds or rewrites a node with the filename of the binary file as the key. By following this pattern, more configurations can be saved in the file.

The backend program does all the needed preparations and fills the statistics, so that the user does not need to add anything on their own – and is strongly advised not to do that anyway, because changing the values in the configuration file can make the storage unreadable.

The configuration file is used by the backend and the web server for passing the bitmap storage statistics. Therefore, the filename passed to the backend and the server must be the same (both are set to “config” by default). The configuration file is saved to custom directory together with the bitmap files.

The bitmap storage statistics allows the server to access the offsets of a particular subnet in the bitmap and visualise binary data in a comprehensive manner. This file is in a YAML [14] format and it defines ranges of both dimensions and their granularities. For each distinct configuration, a new set of binary files is created.

The structure of the configuration file format together with the description of the individual parameters can be seen in Listing 4.2.

```
<filename prefix>:
  addresses:
    granularity: <subnet mask>
    first: <first address>
    last: <last address>
  time:
    first: <timestamp of the oldest interval in time window>
    last: <timestamp of the last processed flow>
    granularity: <time unit in seconds>
    window: <number of intervals stored into the past>
    intervals: <number of intervals passed since the beginning>
  module:
    start: <timestamp of the start of the module>
    end: <timestamp of the end of the module>
```

Listing 4.2: Configuration File Structure

All timestamps are in YYYY-MM-DD HH:MM:SS format, defined for the local time zone.

An example of such created file is then showed in Listing 4.3.

```
mybitmap:
  addresses:
    granularity: 16
    first: 8.0.1.0
    last: 8.3.0.2
  time:
    granularity: 200
    intervals: 150
    window: 100
    first: 2016-04-25 11:04:23
    last: 2016-04-25 13:00:10
  module:
```



```

start: 2016-04-25 11:04:21
end: 2016-04-25 13:00:15

```

Listing 4.3: Configuration File Example

While the backend is still running, time last, module end and intervals values are missing and if the server is started at that point, the frontend considers it an online case. If all presented values are present, frontend handles it as offline.

In C++, the work with `yaml-cpp` [15] is very intuitive. The configuration file is loaded into a node by `YAML::LoadFile(filename)`. Accessing nodes is realized by using the keys as indices. For example, setting the time interval value into the example configuration file 4.3 would look as follows:

```

YAML::Node config_file = YAML::LoadFile(filename);
if (config_file == NULL) {
    // Write an error message
    return 1;
}
config_file["mybitmap"]["time"]["granularity"] = 200;

```

In order to save the changes, modified content of the node has to be streamed into an output stream(`std::ofstream`) representation of the configuration file.

After setting the configuration, 3 bitmap files (for source, destination and source + destination addresses) are created based on the `--filename` parameter. More information about the structure of the bitmap files can be found in Section 4.2.

Parameters Setup

When analysing the incoming flows, information about IP activity is stored by custom subnets. That means comparing IPs and calculating the index of the corresponding subnet.

In order to make the process most efficient, all used IPs are logically shifted to the right so that they only contain bits at indices where the corresponding netmask has a value of “1”.

Calculating the size of the shift:

```

// IP version derived from entered first and last IP address
// size of IP version in bits (32 for IPv4, 128 for IPv6)
int ip_max_prefixlen;

// ip_granularity is set by user
// e.g. /12 == 11111111 11110000 00000000 00000000 for IPv4
int ip_granularity;
...
// 32 - 12 = 20 for IPv4 | 128 - 12 = 116 for IPv6
int shift = ip_max_prefixlen - ip_granularity;

```

Listing 4.4: Shift Size Calculation

Demonstration of shifting:

```
120.55.246.1/20:
  before shifting:
    binary netmask: 11111111.11111111.11110000.00000000
    binary IP:      01111000.00110111.11110110.00000001
  after 12-bit right shift:
    binary IP:      00000000.00000111.10000011.01111111
    decimal IP:      0.          7.          131.         127
```

The shift is performed analogically for IPv6 addresses.

In `ip_activity.cpp`, this functionality is implemented by function `convert_to_granularity (IPAddr_cpp *addr, int granularity)`.

Before the main loop, the bit vector size is calculated. Normally, the size would be equal to:

$$vector_size = \frac{last_address - first_address}{granularity} = \frac{last_address}{granularity} - \frac{first_address}{granularity} \quad (4.1)$$

Since both the first and the last address is already shifted, therefore divided by the granularity, all that needs to be done is the subtraction. This operation is implemented by function `ip_substraction (IPAddr_cpp addr1, IPAddr_cpp addr2)`, where the result of the function equals `addr2 - addr1`.

In the case of IPv4, the subtraction is performed as one operation, using conversion from IPv4 to `uint32_t`, by method `get_ipv4_int()`. However, when the IP is of version 6, the subtraction is divided for the lack of 128b integer type into 4 parts – each IPv6 is by `get_ipv6_int()` represented as a vector of 4 `uint32_t`. The subtraction is performed as an N-base subtraction (in this case, 2^{32} base) as stated in Listing 4.5.

```
// all vectors are of size 4
// ip1 and ip2 have value of corresponding operands
std::vector<uint32_t> ip1, ip2, result;
uint8_t borrow = 0;

// Go from right to left
for (int i = 3; i >= 0; i--) {
    // If the result is positive, perform normally,
    // clear borrow value
    if (ip2[i] >= (ip1[i] + borrow)) {
        result[i] = ip2[i] - ip1[i] - borrow;
        borrow = 0;
    } else {
        // If the result is negative, add up radix
        // (here UINT32_MAX) to achieve a positive result
        // and set borrow to 1
        result[i] = ip2[i] + RADIX - ip1[i] - borrow;
```

```

        borrow = 1;
    }
}

```

Listing 4.5: N-base Substraction

The return value of the function is a 32 bits unsigned integer. If the size of the address space is bigger than `UINT32_MAX`, zero is returned.

Traffic Analysing

In the main loop, the module accesses the source and destination address of each incoming flow. Firstly, the program checks if the current flow belongs to the most recent interval.

If the flow is older, it is stashed and the program continues with the next one.

If the flow belongs to the later interval, the program assumes that from now on, all incoming flows have the `TIME_FIRST` field greater or equal to this one. Therefore, it saves the current interval to the storage and if there are more intervals to be skipped, they are written to the binary file as intervals with zero activity.

When the interval is set up, the program compares the IP version and whether the IPs belong to the address range as follows:

$$first_ip \leq record_ip \leq last_ip \quad (4.2)$$

When calculating the index to the bit vector, the above mentioned functions `convert_to_granularity (analysed_ip, granularity)` and `ip_substraction(first_ip, analysed_ip)` functions are used. The result of the subtraction is then in fact a number of subnets of chosen granularity between the first address in the range and the analysed IP and therefore, does not need further calculations and is directly used to access the bit vector.

Storage and Configuration Update

Every time a flow from later interval is encountered, the current bit vector is passed to function `binary_write(filename, bit_vector, open_mode, index)` to be stored to the storage.

If the current number of processed intervals is lower than the time window, data is appended at the end of the bitmap (therefore, the `open_mode` contains `std::ofstream::app` flag). Otherwise, based on the passed argument `index`, the offset of the new beginning of the bitmap is found (since it is a circular buffer, it changes after each interval) and the current data holding information about the oldest interval are overwritten.

Internally, function goes through the vector and aggregates activity data to bytes, so that it

can be stored to the bitmap (more in Section 4.2). and cleared. The principle is illustrated in pseudocode 4.1.1.

```
// Go through the bit vector
for (uint64_t i = 0; i < vector.size();) {
    // Initialize each byte to 0
    uint8_t byte = 0;

    // Go through the byte from left to right
    // Set individual bits using a mask byte
    for (uint8_t mask = 128; (mask > 0) && (i < size);
        i++, mask >>=1) {
        if (vector[i]) {
            byte |= mask;
        }
    }

    // Stream bytes to file
    // If size of vector is not divisible by 8,
    // the rest of the byte consists of zeros (padding)
    bitmap.write((const char*)&byte, 1);
}
```

After storing new interval to the storage, the number of the oldest interval in the time window and the number of passed intervals has changed. To keep the bitmap readable, these values are updated in the configuration file.

4.1.2 IP Address Data Type

For work with IP addresses, an UniRec class `ipaddr_cpp.h`, which has been extended from its original version in C, is used. Both data types are defined in the Nemea-Framework repository [5]. The IP address itself is stored in a `ip_addr_t` union in big endian. The union is structurized as follows:

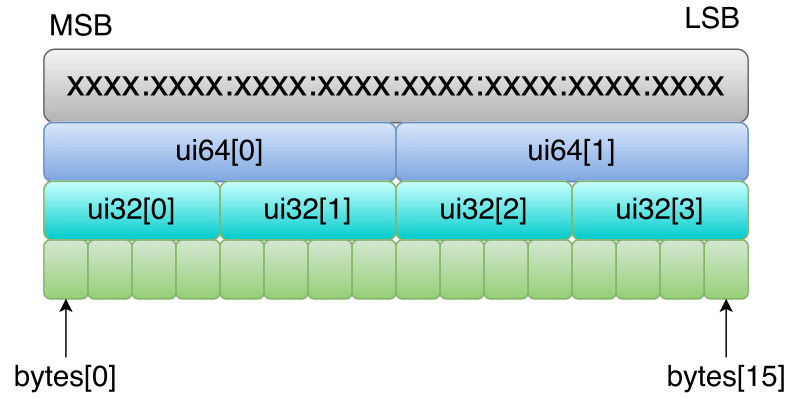


Figure 4.2: ip_addr_t union

Due to the fact that there are two IP versions, IPv4 and IPv6, each of its own size, their location in the `ip_addr_t` is different. Whereas IPv6 simply occupies all 16 bytes, IPv4 is starts at the 9th byte. Internally, the union for IPv4 looks as follows:

```
0000:0000:0000:0000:<ipv4 address>:ffff:ffff
```

4.2 Bitmap Storage

The main storage is a set of 3 files in a binary format, one for each type of address (source addresses, destination addresses, either of the two). It is used for storing big amounts of data efficiently, using one bit for defining state of a given subnet at a given moment. As the result, structure has two dimensions, One row represents an activity of the whole monitored address space in one time interval.

The storage works as a circular buffer. When the storage reaches the time window limit, the oldest data is rewritten. Therefore, when the online mode is chosen, the frontend has to calculate the beginning from the time of the first record as follows:

$$intervals = time_first + intervals_passed \times time_unit \quad (4.3)$$

The bitmap filename uniquely identifies a set of bitmaps and their configuration. The name of each bitmap file is created by appending “_<type>.bmap” to filename. Possible types are defined as follows – s (source address), d (destination address) and sd (source or destination address). It is stored in the same directory as the configuration file.

When storing data, it is more efficient to see a row as one unit of time. It is much easier to concatenate/rewrite one line in a binary file than to go through the file and add one

symbol at the end of each line. Therefore, in the NEMEA module where the speed of data processing is essential, the matrix is structured this way.

Since the size of the bit vector does not need to be divisible by 8, there is a padding at the end of each line to the nearest byte. The padding is the only part of the storage that is not used for storing the activity.

The index for accessing wanted time interval is calculated as follows:

$$offset = interval_position_in_time_window \times CEILING(\frac{vector_size}{8}) \quad (4.4)$$

When accessing data from the web server, the resulting image uses x -axis (columns) for displaying time units, whereas in the storage, it is represented by rows. Therefore, after each query, the server transposes the bitmap before any other specified operation needed for data visualisation.

As for the size of the bitmap file, both maximum size of the bit vector and the time window size are set to 1000 for the option of displaying the bitmap 1:1 conveniently. The limits are set in macros `MAX_WINDOW` and `MAX_VECTOR_SIZE` and can be changed.

Following equations demonstrate the efficiency of the storage and the size of part occupied by padding bits.

Conditions that must be met for all equations:

$$\begin{aligned} intervals &\in \langle 0; time_window \rangle \\ time_window &\leq MAX_VECTOR_SIZE \\ vector_size &\leq MAX_VECTOR_SIZE \end{aligned}$$

Size of the bitmap storage file can be calculated as follows:

$$\begin{aligned} bitmap_size &= size_of_interval \times intervals [B] \\ bitmap_size &= CEILING(\frac{vector_size}{8}) \times intervals [B] \end{aligned}$$

Size of the padding in bitmap storage is expressed as follows:

$$\begin{aligned} padding_size &= size_of_padding_in_vector \times intervals [b] \\ padding_size &= ((8 - (vector_size \bmod 8)) \bmod 8) \times intervals [b] \end{aligned}$$

Based on the size of the padding and the storage itself, let us express what percentage of

the storage is occupied by padding:

$$padding_part = \frac{padding_size}{bitmap_size} \times 100\%$$

$$padding_part = \frac{((8 - (vector_size \bmod 8)) \bmod 8) \times intervals}{CEILING(\frac{vector_size}{8}) \times intervals \times 8} \times 100\%$$

$$padding_part = \frac{((8 - (vector_size \bmod 8)) \bmod 8)}{CEILING(\frac{vector_size}{8}) \times 8} \times 100\%$$

As the equation is reduced, it is clear that the part occupied by padding is influenced by vector size and its divisibility by 8. The graph 4.3 shows the relationship between the vector size on x -axis and percentage of padding in storage on y -axis. The part containing padding bits is getting relatively smaller to the bitmap storage with increasing vector size.

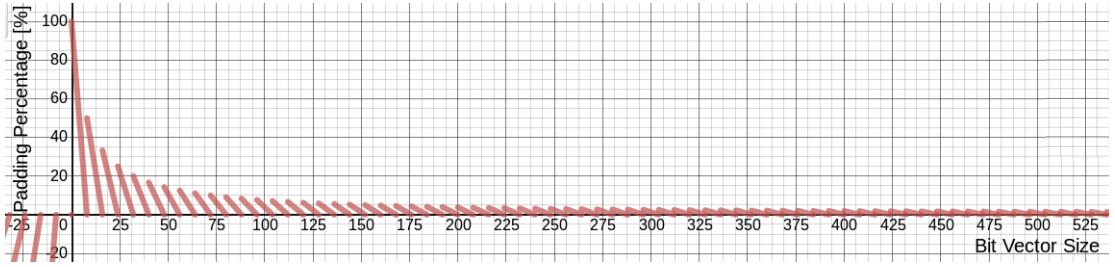


Figure 4.3: Padding percentage based on vector size

Let us take the maximum size of bitmap and maximum padding for illustration, how much of the storage space the padding can occupy at most at default limits. The size of the vector equals 993 for it is the biggest value below MAX_VECTOR_SIZE that has the most (7) padding bits.

$$intervals = 1000$$

$$vector_size = 993$$

$$padding_part = \frac{((8 - (vector_size \bmod 8)) \bmod 8)}{CEILING(\frac{vector_size}{8}) \times 8} \times 100\%$$

$$padding_part = \frac{((8 - (993 \bmod 8)) \bmod 8)}{CEILING(\frac{993}{8}) \times 8} \times 100\%$$

$$padding_part = \frac{((8 - 1) \bmod 8)}{125 \times 8} \times 100\%$$

$$padding_part = \frac{7}{1000} \times 100\% = 0.7\%$$

For 1000 intervals and vector size 993, the percentage is mere 0.7%. The rest 99.3% of the size is occupied by needed data.

When compared to querying data in whole flow records where only one attribute of the whole record is used, the ratio of the stored data to used data here is significantly better.

4.3 Web Server

The web server is implemented in the source file *http_server.py*. For bitmap handling, the server uses the visualisation handler (discussed in Section 4.4).

The server is compatible with both Python 2.x and Python 3.x versions. In order to run the server, several non-standard packages have to be installed for chosen version of Python (see subsection 3.4.2).

The server program creates a new class of a HTTP request handler and passes it the input arguments and the visualisation handler. Then, the program creates a socket, binds it to the defined port and enters an endless loop.

In order to visualise valid data, the server program cannot be initiated until the backend program is activated and the configuration is written in the dedicated file. Also, if backend is run again with a different configuration but the same file names, the server has to be restarted.

4.3.1 Workflow

The main functionality of the server is present in Figure 4.4. The scheme is simplified in order to demonstrate the main operations more clearly.

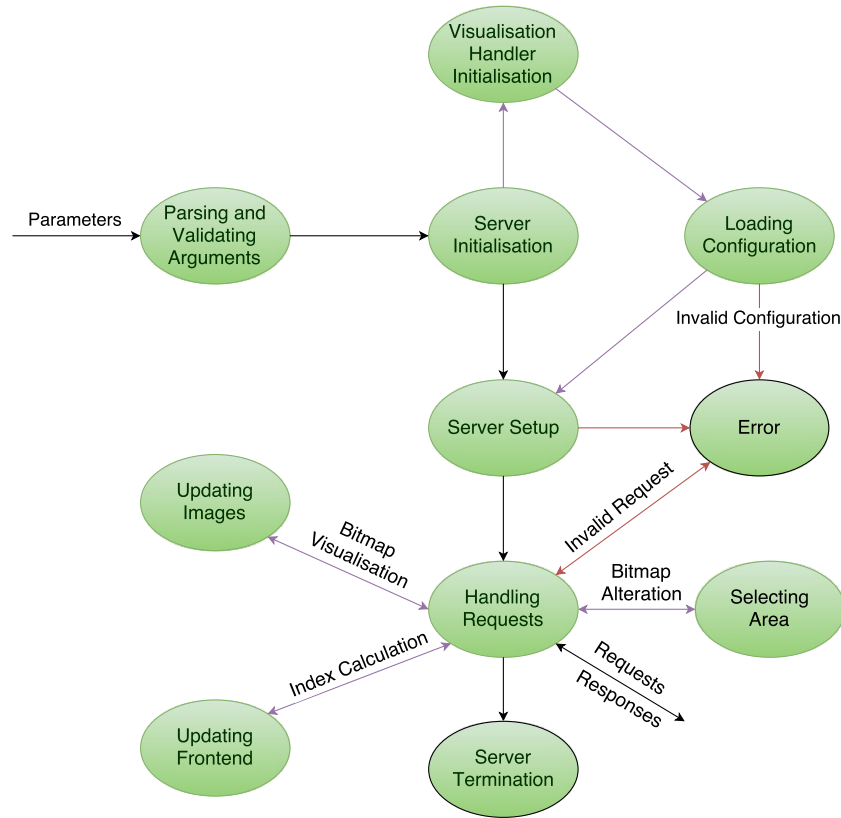


Figure 4.4: Web Server Workflow

The start state is initiated by running the program with custom parameters. The end states are similarly to the backend scheme defined by a black perimeter line. The server can be terminated unsuccessfully before entering serving loop when an error occurs or successfully when an interruption (SIGINT) or terminating (SIGTERM) signal is received. If an error occurs while the server is active, the error message is sent to the standard error output but the server keeps running.

Since the server uses the visualisation handler module, its actions are also added to the scheme, distinguished by purple arrows. Its initialisation takes place right before server setup when the handler object is added to the request handler class as an attribute. Other interactions with the visualisation handler are visible during the request processing where it is the main source of data and enables the server to manipulate it.

One of the states is not addressed directly but discussed in several section. This state is called *Updating Frontend* and it includes sending the main page to the client and calculating the index values of current cursor position.

Parsing and Validating Arguments

At first, the program parses arguments by the user using the `argparse` [1] package. The list of the possible parameters is as follows:

- `-c/--config_file` Filename of the configuration file. (default: "config")
- `-d/--dir` Path to directory with configuration and bitmaps (default: ".")
- `-f/--filename` Filename of bitmap storage files. (default: "bitmap")
- `-H/--hostname` Hostname of the server. (default: "localhost")
- `-p/--port` Server port. (default: 8080)

The user can specify the configuration filename by passing `--config_file`. This approach is needed if the user decided to change the configuration filename in backend. To find the used configuration, the user has to define the filename of the bitmaps using `--filename` parameter that will serve as key in the configuration file to the bitmap parameters.

The server can be run from other directory than the configuration and bitmap files are located in. In that case, the path to the custom directory has to be specified by `--dir` parameter.

If the user wishes to identify the server with custom hostname or port, they can define the `--hostname` or `--port` parameter, bearing in mind the well-known ports [13] (range 0-1023) where they need a root permission.

The received arguments are parsed and validated by the program and then the bitmap parameters are retrieved.

Server setup

The server retrieves bitmap statistics from the configuration file (more about its structure in Section 4.1.1) based on its filename using the `Visualisation_Handler` object. This process is described in Section 4.4.1 where the additional module is described in more detail. After the parameters initialisation, the creation of the request handler is initiated.

In order to be able to handle various requests from the client, the basic request handler class is modified. In function `create_handler(arguments, visualisation_handler)`, the program sets parameters of the bitmap storage based on the configuration file and defines a new class `My_RequestHandler` that is inherited from the `BaseHTTPRequestHandler` class and extends its functionality for its intended use – retrieving and editing bitmap and converting bitmap to image.

Subsequently, the `My_RequestHandler` class together with the defined socket pair `<hostname> :<port>` are used for creating the HTTP server.

Handling Requests

The program then enters an infinite loop where the server listens for incoming requests. All errors occurred during this state can produce an error message but they do not end the loop. The activity of the server can only be stopped by incoming interruption (SIGINT) and termination (SIGTERM) signals.

Since the processed requests are tied up to both the server and the web client, they are discussed separately in Section 4.5.

4.4 Visualisation Handler

The bitmap manipulation used by the server is defined in a separate file called *activity_visualisation.py*. It stores all the bitmap configuration and also the current bitmap for visualisation.

4.4.1 Loading Configuration

In order to fill the handler with correct values, its method `load_config(dir, bitmap_name, config_name)` is called. In this method, the handler stores all the bitmaps and their parameters retrieved from the configuration file using *pyyaml* package. This package uses the same approach as is done in backend – the values are accessed using keys as indexes to the file structure. Beside the configuration file values, similarly to the backend, the handler calculates the size of the address space to be able to index data in the storage.

Then, the handler is called solely for manipulating the bitmap and its statistics. It always stores bitmap internally in ratio 1:1 and scales the image only for the client based on their request.

4.4.2 Functionality

As stated above, the main functionality of the visualisation handler is manipulating the binary storage and retrieving its data. In this part of the thesis, the main operations are introduced.

Index Calculation

One type of operation implemented is getting a data offset from data value and vice versa. This is used for example when calculating the current position of the client's cursor.

The principle of the former operation (offset from value) can be defined as follows:

$$offset = \frac{current_value - min_value}{granularity} \quad (4.5)$$

The latter operation (value from offset) can require additional adjustment due to used units, for example transforming offset to seconds to calculate time. The process can be described followingly:

$$value = min_value + offset_in_units \quad (4.6)$$

Retrieving Bitmap

Other functionality is retrieving bitmap from the storage (implemented in `binary_read(bitmap_name)`). The principle is demonstrated in Figure 4.5 with the size of address space equal to 10, time window 100 intervals and current offset 1 (of the newest interval).

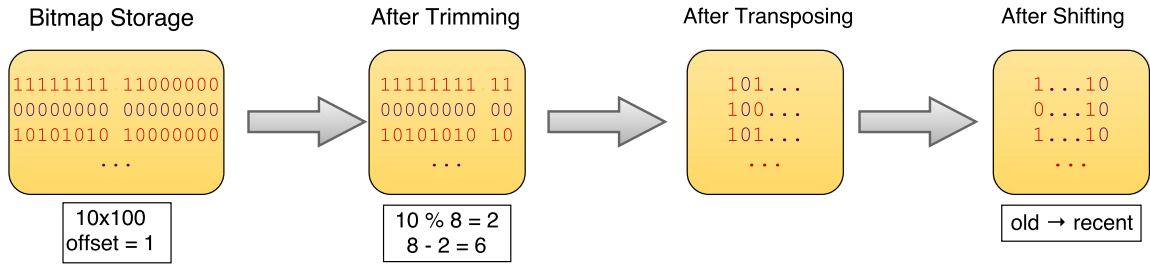


Figure 4.5: Bitmap Retrieval

The handler reads the content of the binary file and trims the padding bits from each row. It then transposes the 2D array so that each row contains information about one subnet. Finally, the handler shifts each row based on the offset of the beginning of the circular buffer to increase readability of the final image.

Creating or Updating Images

Retrieving bitmap data is usually followed by creating the image based on the bitmap (`create_image(bitmap, filename, scaleR, scaleC, height, width, selected)`). The final image always has the same proportions `subnet_size : time_window`. If the whole image is not filled with storage data, the rest is set to an *undefined* value distinguished by gray colour. The original data always covers the top left corner.

In the case of original bitmap, the handler takes the bitmap and scales it in passed ratio – rows by `scaleR:1`, columns by `scaleC:1`. If the bitmap does not cover the whole time window, the rest of the image is going to have *undefined* value.

When creating selected area image, the selected data scaling units are calculated in advance, so that the image is as big as possible without exceeding the size of the original one. When creating the image, only the remaining part of the image, here without scaling, is set to *undefined*.

Passed bitmap is loaded to a 1D buffer. Each bit is then transformed into one pixel – “0” → RGB(0,0,0), “1” → RGB(255,255,255) and *undefined* → RGB(105,105,105). The shade can be customized by changing *undefined* attribute. Created image is then saved to an image file specified by *filename*.

Area Selection

When client wants to select an area of the bitmap, they select the edges of this area – $x_{min}, x_{max}, y_{min}, y_{max}$ and submit the request to the server. The job of the visualisation handler is to crop the original bitmap and leave only the selected area which is implemented in `edit_bitmap(query)` method, leaving only the defined ranges:

```
selected_bitmap = original_bitmap[x_min:x_max][y_min:y_max]
```

The selected bitmap is then scaled in a maximum possible way that does not exceed the size of the original bitmap. The principle is formally defined in Figure 4.4.2.

$$\begin{aligned} \text{let } x_{selected} &= x_{max} - x_{min}; y_{selected} = y_{max} - y_{min}; X, Y \subset \mathbb{N} \\ \forall x \in X \text{ that } x \cdot x_{selected} &\leq address_space \rightarrow x \in X_2 \\ \forall y \in Y \text{ that } y \cdot y_{selected} &\leq time_window \rightarrow y \in Y_2 \\ x_{scale} &= \max(X_2), y_{scale} = \max(Y_2) \end{aligned}$$

Thanks to this scaling, the client can inspect is area of interest more closely – be it IP subnets or specific intervals. The result is sent to the client.

4.5 HTTP Requests and the Corresponding Responses

The server has modified HTTP GET request handler, so that the web client can request various specific data. The other request methods [29] are handled by default handlers in the `BaseHTTPRequestHandler` class. The GET method is handled by class method `do_GET()`. Based on the requested resource (identified by URI [20]), the server generates the response.

The server has implemented behaviour for several types of requests. In this section, the formats of valid requests are shown and the principle of server’s handling is demonstrated.

4.5.1 Main Page

The most basic one is requesting the main page which can be identified as `"/`, `/index.html` or `/frontend.html` as it is physically stored in `frontend.html`. In order to send valuable information back to the client containing the statistics of the bitmap file, the server modifies the contents of the `frontend.html` file, adding current storage information to the *Characteristics* section. Using the package *BeautifulSoup*, the file is parsed and the information is formatted and appended to the designated elements.

4.5.2 Updating Image

Another type of request is used for updating the image. When the online mode is active, the client sends the asynchronous update request periodically. The update interval can be changed in the frontend file `frontend.js` by setting `update_interval` value and is set to 30 seconds by default. In the case of offline mode, client sends update request only when the page is refreshed. In both modes, the server loads the new storage contents (provided they exist), visualises the data in defined scaling and sends an encoded image back to the client.

The URI of the request for updating the image looks as follows:

```
GET /<path to image>?update=true&scale=<ratio> HTTP/<version>
```

The request consists of the path to the image that is consistent – `"images/image_<bitmap type>.png"`, so that the bitmap type does not need to be specified separately. To indicate this type of the request, the type update with symbolic value is added to the URI. Since the user can also set scaling of the image, it also specified in the URI format.

The response to this request depends on if the requested bitmap exists. If it does not, the server responds with a response header of the following format:

```
HTTP/<version> 404 Not Found
Bitmap: none
```

If the bitmap does exist, after creating the image of specified characteristics, the following response is sent back:

```
HTTP/<version> 200 OK
Content-Type: image/png
Interval_range: <range>
Mode: <online/offline>
Bitmap: ok
```

```
<base64 encoded image>
```

In this response, the number of displayed interval range is sent together with the current mode, so that the client can react to the change from online mode to offline. `Bitmap: ok` indicates that the client can decode the received response and display the result to the user.

4.5.3 Area Selection

The next request is sent when only a specific area of the image is to be displayed in a dedicated element. The server creates a new image from the original bitmap based on the query values, scales it (principle demonstrated in formal definition in Figure 4.4.2) and sends it back to the client in the response.

The request has the following format:

```
GET /?select_area=true&bitmap_type=<type>&
    first_ip=<ip>&last_ip=<ip>&
    first_time=<time>&last_time=<time> HTTP/<version>
```

The parameters consist of the limits of the specified area and the bitmap type.

If the original bitmap does not exist, the server sends only a header informing that no bitmap is available:

```
HTTP/<version> 404 Not Found
Bitmap: none
```

If the bitmap does exist, the server sends the encoded image visualising the specified area together with the scaling ratios of both dimensions as demonstrated here:

```
HTTP/<version> 200 OK
Content-Type: image/png
IP_unit: <scaling of IP subnets>
Time_unit: <scaling of time intervals>
Bitmap: ok

<base64 encoded image>
```

4.5.4 Index Calculation

This request type is used for finding the IP subnet and time interval value at a given index. When the user hovers over the image, the IP subnet and the interval is displayed in the dedicated table **Current Position**. Since JavaScript does not have any convenient library that would add up an IP address (possibly an IPv6 in the reduced form) and an integer, it is easier to get the value from the server that is able to implement this functionality using *ipaddress* package. And while doing that, it also calculates the interval index.

The position is calculated for currently displayed image, so that the frontend only needs to specify whether the operation is to be carried out on the original bitmap or the selected area. Related calculation methods are discussed in Section 4.4.2.

The request format looks as follows:

```
GET /?calculate_index=true&bitmap_type=<type>&
    first_ip=<ip>&ip_index=<index>&
```

```
first_time=<time>&time_index=<index> HTTP/<version>
```

This format is distinguished by the type `calculate_index`. It contains the bitmap type that specifies if the current position is to be calculated for the original bitmap or the selected area. Besides that, the first values in both ranges altogether with current indices are inserted and passed to the server methods responsible for the calculation.

After computing the values, the server sends the following response to the client:

```
HTTP/<version> 200 OK
Content-Type: text/plain
IP_index:<calculated IP at index>
Cell_colour:<colour of the current position>
Time_index:<calculated time at index>
```

The response is in the plain text format and it consists of both IP and time values at the specified indices. The values are sent in headers `IP_index` and `Time_index`. If the user hovers over an area with undefined values, both values equal “undefined”.

In order to illustrate the current position activity status, the colour representing the activity at specified coordinates is sent as a string (“white” for active, “black” for inactive or “gray” for undefined).

4.6 Web Client

The web client is implemented in source files with the *frontend* filename. It consists of a HTML file with the basic page content, CSS stylesheet and JavaScript file describing the page behaviour. The webpage layout is discussed further in Section 3.2.3.

The client displays the monitored activity to the user. All the data is supplied to the frontend by the server in the form of a HTTP response.

4.6.1 Page Content

The main page content is defined in *frontend.html* and used stylesheet in *frontend.css*. Since the displayed information changes based on monitored traffic, some of the elements are added or modified dynamically by JavaScript.

Original Bitmap

The top part of the main page is displayed in Figure 4.6 and is dedicated to the visualisation of the whole bitmap. It is divided into several areas. The left column is dedicated to data statistics and the right one is used for modifying the image.

IP Activity analysis

[Home](#)
[About](#)

Original Bitmap

Characteristics

| | |
|--------------------|--|
| Mode | offline |
| Subnet Size | /8 |
| IP Range | 130.0.0.0 - 255.0.0.0 |
| Time Range | 2016-04-24 12:55:58 2016-04-24 12:59:22 |
| Intervals in Total | 666 |
| Visible Intervals | 20 |
| Time Interval | 10 seconds |
| Time Window | 20 intervals |

| Current Position | |
|------------------|---------------------|
| IP Subnet | Time |
| 130.0.0.0/8 | 2016-04-24 12:55:58 |

Type

Source IPs

Destination IPs

Both

Scale

1:1

2:1

Select Bitmap Area

| | |
|------------|---------------------|
| IP Range | 130.0.0.0 |
| | 255.0.0.0 |
| Time Range | 2016-04-24 12:55:58 |
| | 2016-04-24 12:59:22 |

Submit

Copyright © 2016 CESNET

Author: Katerina Pilatova, FIT VUT

Figure 4.6: Frontend Main Page

The **Characteristics** area contains bitmap statistics and its purpose is to improve the readability of the visualised activity. The user has better idea about the units in both dimensions and knows which data are currently displayed.

The full list of the characteristics and their brief description can be found below:

- Mode Mode the activity monitoring is in. Value: *online* or *offline*.
- Subnet Size Size of the network mask dividing monitored address space.
- IP Range Contains monitored IP range. Value: *<first IP>* - *<last IP>*
- Time Range Contains visible interval range as timestamps.
- Intervals in Total Total number of scanned intervals by backend module.
- Visible Intervals Number of currently visible intervals. Value: 0 – *time_window*
- Time Interval Size of one time interval in seconds.

- Time Window Size of the time window in intervals.

Below the characteristics is located the **Current Position** which changes dynamically as the user's cursor moves over the image area. The background reflects the activity status of the current coordinates and the exact values of the subnet and time interval coordinates are also included. If user hovers over the area with undefined values, corresponding values are set to *undefined*.

The user can change the displayed bitmap type (section **Type**) and scale image so that the smaller areas are more visible (section **Scale**). The user can also select area they would like to examine in detail by changing the IP or time range values in the form and submitting the form. After submitting the form, a previously hidden area with information about the specified ranges is now visible.

Under the statistics and options, the visualised activity is displayed as an image. Black areas indicate inactivity, white areas active subnets and gray areas undefined values. The interface enables the user to interact with the image.

The user can hover over the image to display activity of a certain position or select area by clicking on a chosen position and dragging pointer to the chosen range limit, creating a frame around chosen area in the process (see Figure 4.7. The user can drag the pointer in any direction. If the user does not drag in left-to-right top-to-bottom direction, the values are automatically swapped.

| | |
|--------------------|--|
| Mode | offline |
| Subnet Size | /8 |
| IP Range | 120.0.0.0 - 220.0.0.0 |
| Time Range | 2016-05-15 20:01:55 2016-05-15 20:11:50 |
| Intervals in Total | 32 |
| Visible Intervals | 32 |
| Time Interval | 20 seconds |
| Time Window | 50 intervals |

| Current Position | |
|------------------|-----------|
| IP Subnet | Time |
| 186.0.0.0/8 | undefined |

Source IPs Destination IPs Both

Scale
1:1 2:1

Select Bitmap Area

IP Range

153.0.0.0

182.0.0.0

Time Range

2016-05-15 20:01:55

2016-05-15 20:11:50

Submit

Figure 4.7: Selecting area by dragging cursor

When the area is specified, the form values of both ranges are updated with chosen limits values (see Figure 4.8). The changed area is highlighted. If one dimension value equals *undefined*, maximum defined value in that range is used. In case both values are *undefined*, the full range of both dimensions is inserted.

| | |
|--------------------|--|
| Mode | offline |
| Subnet Size | /8 |
| IP Range | 120.0.0.0 - 220.0.0.0 |
| Time Range | 2016-05-15 20:01:55 2016-05-15 20:11:50 |
| Intervals in Total | 32 |
| Visible Intervals | 32 |
| Time Interval | 20 seconds |
| Time Window | 50 intervals |

| Current Position | |
|------------------|-----------|
| IP Subnet | Time |
| 144.0.0.0/8 | undefined |

Source IPs Destination IPs Both

Scale
1:1 2:1

Select Bitmap Area

IP Range
163.0.0.0
186.0.0.0

Time Range
2016-05-15 20:10:35
2016-05-15 20:11:50

Submit




Figure 4.8: After releasing the cursor, form is updated

After submitting the form, the **Selected Area** section is displayed, containing specified area of the original image and its statistics (see Figure 4.9). The form values specifying the selected area are inserted in the characteristics.

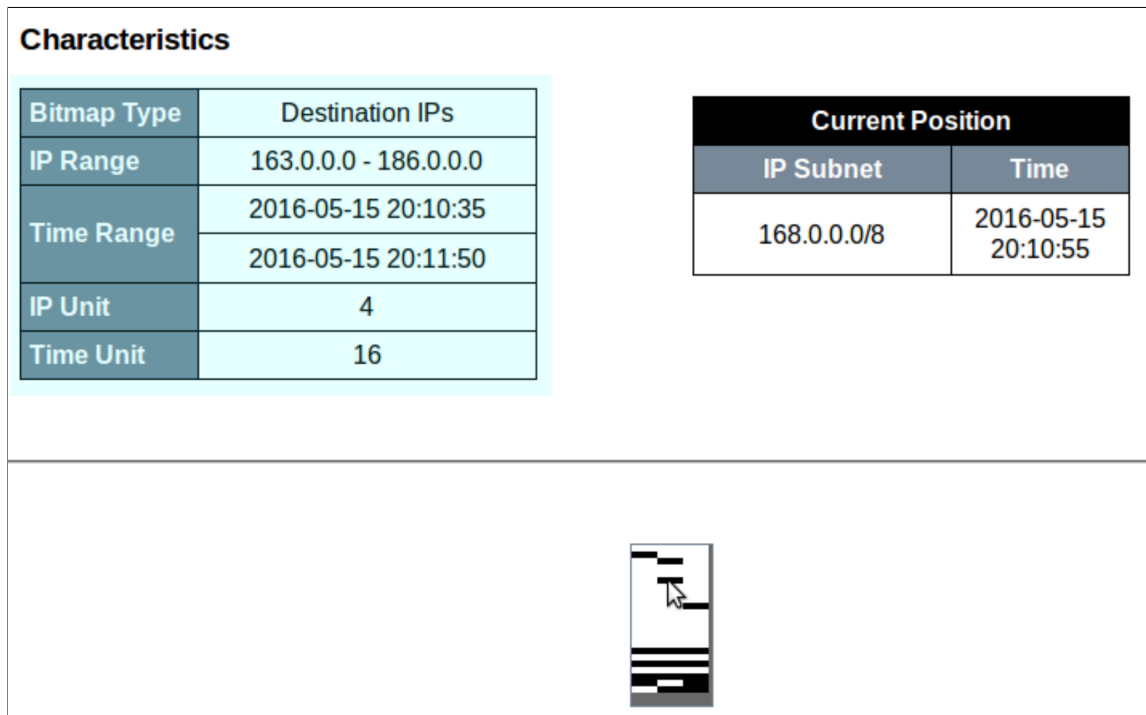


Figure 4.9: After submitting the form, Selected Area is displayed

Selected Area

In the selected area, no modification options are available. The left column is again dedicated to the **Characteristics**. This time, only the attributes that are different from the main image are present. Moreover, new attributes are added. Bitmap type defines from which storage the area was created. IP and time interval units define the scaling of the image in both dimensions (for example, IP unit with value 8 means that activity of one subnet is displayed on 8 pixels on the y -axis).

Full list of displayed characteristics:

- **Bitmap Type** Mode the activity monitoring is in. Value: *online* or *offline*.
- **IP Range** Contains selected IP range. Value: *<first IP>* - *<last IP>*
- **Time Range** Contains selected interval range as timestamps.
- **IP Unit** Defines scaling for subnets. One subnet is on *<value>* pixels.
- **Time Unit** Defines scaling for intervals. One interval is on *<value>* pixels.

The right column contains **Current Position** which works analogically to the top version. The two position indicators work independently.

The visualisation of the selected area below the characteristics only enables the user to hover over, no additional selecting is available.

4.6.2 Client Interactivity

In order to be able to change parts of the page dynamically without refreshing the main page, a JavaScript file *frontend.js* is included. This script realises all the interaction with the web server and reacts to server's responses. In this section, the main techniques and event handlers are described.

Since the script uses the *jQuery* library for easier element access and modification, the script is not interpreted until the page is fully loaded.

Initialisation

When the page is fully loaded, it contains all the characteristics and default values from the server but the image initialisation is up to the client.

If the user is currently in online mode, the top image is to be updated periodically. Therefore, a timeout handler is initialised with the default value of 30 seconds. The automatic update is implemented in the function `auto_update()`. The update cycle is started by immediately calling the `update_bitmap()` function. In the offline mode, the image is updated only once at the beginning.

In addition to the image update, **Current Position** values and form values are set to default (typically range limit values) and all dynamic variables needed for the interactivity are set (such as mouse index or IP version).

HTTP Requests

All HTTP GET requests and the corresponding responses are handled by function `http_GET(-url, callback, arguments, content_type)` using AJAX.

Upon calling this function, a new request is created from passed `url` and `arguments` creating the query. The callback function is set to be called upon the server's response.

Bitmap Update

When the image is to be updated, the current version has to be obtained from the server. This is done by `update_bitmap()` function which sends a HTTP GET request to the server, using the `http_GET` handler and as a callback function, `set_bitmap()` is passed to the handler.

When the response is received, the callback function checks if the requested bitmap was available. If yes, it creates a new `` element in designated block element with ID `bitmap_inner`, containing the server's encoded response.

In addition to displaying the image, the server also sends new statistics values in the response header. Thanks to this, the client can keep the user up to date with the total number of intervals and the time range of the displayed activity in online mode.

Form Submit

When submitting the form, the usual action would be sending the values as a query to the server and reloading the page. Here, if the page were reloaded, it would lose all the data from the last session, such as last visited position. In order to avoid this, a custom handler is implemented for click event for `submit` class elements.

This handler disables the default action, parses and validates the form values by comparing the selected range limits to the full bitmap limits.

Because JavaScript does not have any convenient comparison function for IP addresses, a custom function `compare_ips(first_ip, last_ip)` is used. If reduced IPv6 addresses are entered, they are normalized at first. Then, the two addresses are split into parts based on the delimiter ("." in IPv4 and ":" in IPv6) and each part is compared separately.

After arguments validation, an HTTP GET request is created. Upon the server's response, a callback function `set_selected_area()` is called. Here, the Selected Area section is set to visible, the area characteristics is inserted into designated elements and the current position is initialised to default values.

Current Position Update

When hovering over the activity visualising images, a custom `mousemove` handler is implemented. It calculates the cursor position relatively to the image top left corner and subtracts the size of the image border.

Subsequently, it sends a request to the server to get the current position values, providing the underlying image (original or selected area), first values in range and the current ones in the query.

When the server returns the current values, The corresponding **Current Position** section is updated and the element background is set based on the activity status.

Interactive Area Select

In order to spare the user from having to read the limit ranges of their specified area and insert them to the form manually, there is a better option available. The user can drag the

mouse over the area they want to examine and the values are automatically inserted in the form.

This functionality uses the current position indicator for converting the indices to IP and time values. To implement this functionality, three event handlers must be implemented – `mousedown` initialising the drag, `mousemove` and `mouseup` finishing the drag and updating the form values.

During the `mousedown` event, the current position is saved and a block element of a distinctive colour is made visible for the user to know which area they have chosen so far.

This event is followed by `mousemove`, where the block over the area is changed to adjust the user's selection.

Finally, `mouseup` event is registered. To enable choosing the edge positions, the `mouseup` can be carried out anywhere on the page but only if the drag flag is active, the handler is activated. In this handler, the values are inserted in the form in such order that they respect the left-to-right, top-to-bottom orientation. By doing so, the user has more flexibility when creating the area and can move the mouse in all directions.

Additionally, the area selection enables the user to use the *undefined* areas. When only one of the two dimension values is undefined, the maximum defined value is taken. By using this principle, the user does not need to worry about dragging the cursor too far and can use this functionality to display the whole interval for a chosen subnet when the current number of visualised intervals is too small for accurate in-place selection.

Chapter 5

Testing

During the system development, the repeated manual testing of all the main parts of the system was required (the backend, web server and the web client). However thorough, the manual testing is not sufficient for being the only way of testing the system that is under development and frequently changed.

In order to test the most crucial and frequent operations on various data, unit tests were created. These tests are possible due to the system's modularity. However, they do not cover everything, only the parts of the functionality that can be isolated. These tests can be found in the source code attached to this thesis.

User testing is required to test the rest of the functionality. The system was introduced to the consultant and the supervisor and their feedback was taken into account and helped to improve some of the principles and the user interface. This remains an ongoing process as more users start using this system and more feedback is expected in the future.

5.1 Manual Testing

Since the system contains custom data storage and all its parts are interacting with each other, the development of the application started with the backend which is responsible for monitoring and storing the activity to ensure the validity of the storage data. The development continued with the server retrieving and manipulating the stored data to make sure that the data is read and visualised correctly. This was soon followed by the interactive frontend where it had been ensured that the activity characteristics and the images were displayed correctly and the user was given valid information when interacting with the interface.

Manual testing of the backend and the server was realized using the standard and error console output, using debug messages. When debugging the frontend, the most efficient way to test it during the running time was to write the required status updates in the browser console log.

5.2 Unit Tests

In addition to the manual testing, unit tests were written for the backend and the server. These use the functionality from the backend accessible from backend header file *ip_activity.hpp* and the visualisation handler in *activity_visualisation.py*.

They test various input parameter combinations, and more importantly, check if the bitmap in the backend that is stored in the binary storage equals the bitmap retrieved by the server and manipulation with the bitmap on the server.

Backend parameters testing can be found in *backend_tests.sh* and server parameters testing in *server_tests.sh*. Testing the storage content is in *bitmap_tests.sh* and in case of failure, offers an output showing the differences. Configuration file testing can be found in *config_tests.sh* and it tests the backend's creation and server's loading of the configuration file. In order to enable running all tests, *run_tests.sh* was created.

All tests can be run with a verbose parameter *-v* which displays error message in case of failure. It also prints configuration file content and binary storage content for better demonstration. In case of bitmap tests, the results differ due to bitmap transposition by the server.

The functionality of the server based on the incoming requests is not suited for unit testing as it should not be standalone and not based on networking.

5.3 User Testing

This type of testing was realized in several rounds. The participants were introduced to the main principles and the frontend and asked to suggest improvements.

5.3.1 The Backend and the Web Server

In the first round, the main goal was to find out if the data processing and storage maintenance principles were sufficient.

The following suggestions were made:

1. The system should not be based on SIGALRM due to offline mode where the flows are processed in one go. The field *TIME_FIRST* should be used for this instead.
2. The code should be more structuralized in order to be able to call the storage functions separately and test them without having to start the system.
3. The server should obtain the beginning of the oldest visible interval after each bitmap update in addition to the passed number of intervals so that in the online mode, this value can be displayed to the user.

Based on the feedback, the following improvements were made:

1. In both offline and online mode, the system filters the traffic based on the `TIME_FIRST` of the flow. This value is also used for finding out if the current flow belongs to the current interval or not which enables older flows to be stashed and inactivity to be shown when skipping multiple intervals. The configuration file has a different structure for each mode so that the server can distinguish them.
2. In order to make unit testing possible, the backend header file and a new server module were created, which was dedicated solely to manipulate with the bitmap and its characteristics.
3. After each interval, the configuration file is updated and the oldest interval's `TIME_FIRST` is stored, along with the total number of intervals.

5.3.2 Web Client

In the second round, the aim was to test the frontend and find out what information the user wants to see directly on the page and what the image should look like. The feedback was as follows:

1. The 1:1 ratio is not enough for the user to distinguish small areas of a few pixels.
2. Since the image includes only passed intervals, the user does not have such a clear idea about the full size of the image and the proportions when the intervals shown have not reached the time window.
3. The selected area should be scalable based on which dimension the user wants to examine more in detail.
4. The URL does not change based on the requests made which does not allow the user to use the URL to access a specific data. This could be a useful improvement.

The feedback was evaluated and the following improvements were made to the system:

1. The original bitmap has now the option of being displayed in 2:1 ratio. The scaling functionality is implemented for any X:1 ratio so that custom ratios can be added in the future.
2. The images always keep the full proportions. An *undefined* value is added so that when the user displays the image with a smaller number of intervals, the rest of the image has a distinctive appearance to show that the activity has yet to be determined.
3. The selected area also retains the size and proportions of the original image. In order to enable the user to focus on a specific dimension, the principle of selecting area scales both dimensions maximally, but does not allow the full size to be exceeded. By using this technique, the smaller the size of the dimension selected, the more it is scaled.

4. Since this suggestion does not address a crucial functionality, it does not have such a high priority. However, this improvement could indeed extend the possible ways of interacting with the frontend so this functionality is intended to be added in the future.

5.4 Results

The main purposes of this thesis are to reduce storage size needed for storing data containing IP address activity and by doing so, reduce time needed for data lookup in a bigger volume of data using gained information about the activity. In the following sections, the performance, storage and lookup time comparison will be discussed.

5.4.1 Performance

The throughput of a probe during the peak hours equals 160 000 flows per second. In the developed system, the backend is able to process up to 550 000 flows per second in the offline mode. Therefore it has no problems with 30 000 flows per second measured during average traffic hours.

5.4.2 Storage Size Comparison

In order to compare storage size, let us assume the following parameters:

- The target address space is a /16 subnet of IPv4 addresses.
- The granularity equals /32 (individual addresses).
- The interval length is 5 minutes.
- The time during which the activity is to be monitored is one day.

In the case of unoptimized storage, stored flow records that were captured by 10 probes during one day have the total size of 200 GB.

When using the developed system, the size of the storage space is as follows:

$$\begin{aligned}
address_space &= \frac{2^{16}}{2^{32-0}} = 2^{16} \text{ addresses} \\
interval_length &= 300 \text{ seconds} \\
seconds_in_day &= 86400 \\
time_window &= \frac{seconds_in_day}{interval_length} = \frac{86400}{300} = 288 \text{ intervals} \\
storage_size &= CEIL(\frac{address_space}{8}) \cdot time_window \text{ B} \\
storage_size &= CEIL(\frac{2^{16}}{2^3}) \cdot 288 = 2^{13} \cdot 288 = 8192 \cdot 288 \text{ B} \\
storage_size &= 2,359,296 \text{ B} \approx 2359.30 \text{ kB} \approx 2.36 \text{ MB}
\end{aligned}$$

Since the system stores information about three types of addresses, the total size of stored data would three times this size, or:

$$2,359,296 \cdot 3 = 7,077,888 \text{ B} \approx 7.08 \text{ MB} \quad (5.1)$$

The difference between the storage containing flow records and solely IP activity is then defined as follows:

$$\frac{7,077,888}{200 \cdot 10^9} \cdot 100\% = \frac{1728}{48,828,125} \cdot 100\% \approx \frac{1}{28257} \cdot 100\% \approx 0.0035\% \quad (5.2)$$

From the result can be concluded that the developed system uses in this case 28257× less storage space than the ordinary system. If the system were to maintain 200 GB of data using the same parameters, it would contain activity during 28 257 days which is approximately 77 years.

The only data that is not used for storing activity in this system are the padding bits rounding up the row size to the nearest Byte. This matter is discussed in more detail in Section 4.2.

5.4.3 Lookup Time Comparison

Assuming the data parameters from the storage comparison in Section 5.4.2, even with no indexing involved, generally speaking, going through 200 GB data takes much more time than going through mere 2.3 MB.

When using *nfdump* to process the unoptimised storage, going through the whole storage in order to analyse activity of one IP address during the whole time period (one day) took 40 minutes. As *nfdump* did not have any information on the flows, it had to search all present flow records, one by one.

In this system, querying data in the used binary storage is much more effective. In order to get an index for a particular IP address or interval, easy calculation is executed by the server (discussed in Section 4.4.2) and as a result, offsets of both dimensions are obtained.

So, in order to find out IP activity for one day, the index of the address is obtained based on the first address in range, address granularity and the wanted address. After having the offset, only <number of intervals> values are analysed. When processing by the server, it transposes the dimensions and shifts the storage by the offset of the beginning, so that activity of one address is in one row in chronological order. As a result, user can determine the active intervals easily.

Chapter 6

Conclusion

The volume of flow records transmitted through the network is growing. In order to be able to query data faster, an effective way of storing the data and indexing them is required. This thesis addresses this problem and particularly focuses on storing and visualising activity of communicating IP addresses. The IP activity can be defined by a binary value which reduces the total size of the storage.

The aim of this thesis was to develop a system that monitors, stores and visualises IP activity. This system contains a backend, a NEMEA module, which scans incoming traffic and stores the relevant activity to binary storage and its characteristics to a configuration file. The fact that this storage only uses one bit to represent activity of one subnet during one interval effectively reduces the the lookup time.

The stored data is then used by the web server to provide the visualised form of the activity and its characteristics to the user via a web interface. The server uses a separate module for bitmap manipulation and data gathering.

When comparing the results with *nfdump* tool in Section 5.4, it was established that the developed system enables storing IP activity in a more effective way than it is currently done – together with other attributes contained in the flow records. The significantly reduced storage size and easily performed indexing also make querying data much faster than by analysing all records one by one and offers a way of visualising the activity to the user in a comprehensive manner.

Additionally, the system enables the user to display areas of interest in more detail. The gained activity information can then be used to narrow the amount of data processed in other networking tools, such as NfSen.

Currently, the system is in a state that is usable but not final. As this system is deployed and used by various types of users, new requirements for the system can be obtained. The currently planned improvements include to:

1. Integrate the results of this work with NfSen, with distributed collector and fdist-dump [4].

2. Enable the user to interact with the server not only via the web interface, but also by sending queries directly from the browser.
3. Add a system for applying custom ratios to both images (the original and the selected area).

Bibliography

- [1] argparse – Parser for command-line options, arguments and sub-commands.
<https://docs.python.org/3/library/argparse.html>. [Online; Accessed 2016-04-20].
- [2] Beautiful Soup Documentation.
<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>. [Online; Accessed 2016-04-01].
- [3] Boost C++ Libraries. <https://www.boost.org/>. [Online; Accessed 2016-04-25].
- [4] CESNET/fdistdump: Tool for distributed querying of flow data.
<https://github.com/CESNET/fdistdump>. [Online; Accessed 2016-04-25].
- [5] CESNET/Nemea-Framework. <https://github.com/CESNET/Nemea-Framework/>. [Online; Accessed 2016-03-01].
- [6] ipaddress – IPv4/IPv6 manipulation library.
<https://docs.python.org/3/library/ipaddress.html>. [Online; Accessed 2016-04-20].
- [7] jQuery. <https://jquery.com/>. [Online; Accessed 2016-04-01].
- [8] NFDUMP. <http://nfdump.sourceforge.net/>. [Online; Accessed 2016-02-10].
- [9] NfSen – Netflow Sensor. <http://nfsen.sourceforge.net/>. [Online; Accessed 2016-02-05].
- [10] Pillow – Pillow (PIL Fork) 3.1.2 documentation.
<http://pillow.readthedocs.io/en/3.1.x/index.html>. [Online; Accessed 2016-04-20].
- [11] Python Imaging Library (PIL). <http://www.pythonware.com/products/pil/>. [Online; Accessed 2016-02-01].
- [12] PyYAML Documentation. <http://pyyaml.org/wiki/PyYAMLDocumentation>. [Online; Accessed 2016-04-20].
- [13] Service Name and Transport Protocol Port Number Registry.
<http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>. [Online; Accessed 2016-04-20].

- [14] YAML Ain't Markup Language. <http://www.yaml.org/start.html>. [Online; Accessed 2016-04-01].
- [15] yaml-cpp. <https://github.com/jbeder/yaml-cpp>. [Online; Accessed 2016-04-01].
- [16] NetFlow Definition. <http://www.caligare.com/netflow/netflow.php>, 2003-2015. [Online; Accessed 2016-01-25].
- [17] GitHub - CESNET/ipfixcol. <https://github.com/CESNET/ipfixcol/tree/master/plugins/storage/unirec>, 2015. [Online; Accessed 2016-01-25].
- [18] BaseHTTPServer – Basic HTTP Server – Python 2.7.11 documentation. <https://docs.python.org/2/library/basehttpserver.html#module-BaseHTTPServer>, 2016. [Online; Accessed 2016-02-01].
- [19] GitHub - CESNET/Nemea-Framework/libtrap. <https://github.com/CESNET/Nemea-Framework/tree/master/libtrap>, 2016. [Online; Accessed 2016-01-25].
- [20] T. Berners-Lee and R. Fielding. RFC 3986 – Uniform Resource Identifier(URI): Generic Syntax. Internet Requests for Comments, January 2005. (DOI) 10.17487/RFC3986.
- [21] R. Gerhards. RFC 5424 - The Syslog Protocol. Internet Requests for Comments, March 2009. (DOI) 10.17487/RFC5424.
- [22] P. Matoušek. Opora předmětu ISA, kapitola 10: Správa sítě. FIT VUT, 2013.
- [23] C. Muelder and T. Bartoletti. *Interactive Visualization for Network and Port Scan Detection*. University of California, Davis. Available from: <http://web.cs.ucdavis.edu/~ma/papers/raid05.pdf>.
- [24] Mills N. Brownlee. RFC 2722 - Traffic Flow Measurement:Architecture. Internet Requests for Comments, October 1999. (DOI) 10.17487/RFC2722.
- [25] K. Pilátová. Tlacenka/Nemea-Framework. <https://github.com/Tlacenka/Nemea-Framework/>. [Online; Accessed 2016-05-16].
- [26] K. Pilátová. Tlacenka/Nemea-Modules. <https://github.com/Tlacenka/Nemea-Modules>. [Online; Accessed 2016-05-16].
- [27] J. Postel. RFC 777, 792 - internet control message protocol. Internet Requests for Comments, September 1981. (DOI) 10.17487/RFC0792.
- [28] J. Quittek and T. Zseby. RFC 3917 - Requirements for IP Flow Information Export (IPFIX). Internet Requests for Comments, October 2004. (DOI) 10.17487/RFC3917.
- [29] R.Fielding and J. Reschke. RFC 7231 – Hypertext Transfer Protocol(HTTP/1.1): Semantics and Content. Internet Requests for Comments, June 2014. (ISSN) 2070-1721.
- [30] M. Žádník V. Bartoš. *NEMEA Framework for stream-wise analysis of network traffic*. Technical report, CESNET, September 2008. Available from: <https://www.cesnet.cz/wp-content/uploads/2014/02/trapnemea.pdf>.

Appendices

List of Appendices

A CD Content 57

B Installation Guide 58

 B.1 Requirements 58

 B.2 Additional Packages 58

 B.3 Running the System 59

Appendix A

CD Content

This section is concerned with the contents of the attached CD. It includes the basic set of files forming the developed system.

However, not all files needed for running the system are included – the autotool configuration file and IP address library for C++ which was extended for the purposes of this thesis are available in the git repository forks (see Section [B.1](#)).

The main sections are:

- . The root directory contains source files, scripts and guides.
- images/ In this directory, visualised images are to be stored.
- tests/ This directory is dedicated to testing and contains test suite.
- doc/ Here the the source files for the thesis text version are located.

Even though most of the files contain a section with the used licence, *licence.txt* file is also included.

Appendix B

Installation Guide

B.1 Requirements

In order to use the system, the following requirements must be met:

- The target system must have g++ installed.
- The target system must have Python 2 or 3 available.
- In order to run the web client, the target browser must have JavaScript enabled.
- In order to install the backend, access to Nemea repository is required.
- It is essential to download the updates of the Nemea repository and Nemea framework from my forked repository of Nemea-Framework [\[25\]](#) and Nemea-Modules [\[26\]](#). In there, changes of autotools configuration file and *ipaddr_cpp.h* are up-to-date.

B.2 Additional Packages

- For backend, only C++ *Boost* library and *yaml-cpp* version 0.5.3 libraries are required. In order to run the backend on a machine (for example *benefizio*) where the 0.5.3 version is not available, the library has to be installed straight from the source [\[15\]](#) and the path to it included in the *Makefile.am*.
- The attached archive contains a script *install_dependencies.sh* which should install the required packages for both C++ (if the 0.5.3 version is available) and Python both 2.x and 3.x.

B.3 Running the System

The backend has to be run first so that it can create the bitmaps and more importantly, the configuration file. The backend parameters are discussed in Section 4.1.1 and can be displayed by entering `-h [trap]` parameter.

The server program cannot be executed until the configuration file is created. Server's parameters are listed in Section 4.3.1 or can be displayed by entering `-h` or `--help` parameter. When executing the server, make sure to set up the port based on your options.

The frontend program has to run in a browser which has *JavaScript* enabled. It does not need an internet connection as the *jQuery* library is provided in the archive.