Resource estimation of upper layer network traffic analysis

Birgir Haraldsson

Supervisor: Adam Bridgen

Faculty of Information Technology,
University of Akureyri

Submitted April 2005, in partial fulfilment of
the conditions of the award of the degree BSc.

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature _____

Date ____ /____ /_____

Abstract

This paper describes an experiment to create estimates of how much resources are needed to analyze network traffic at the upper network layers. The paper shows that upper layer analysis requires significantly more resources than a lower layer analysis. Upper layer analysis introduces many issues including concerns for privacy. A prototype software to filter at the upper layers was implemented using multi-threaded pipeline of filters. The prototype was further used in a sequence of experiments, with varying loads and searches, to generate resource estimates.

Contents

Chapter 1

Introduction

Network traffic analysis tools are one of the many things that are evolving quite fast due to increased request for such tools. These tools are being used for scanning for viruses, attacks and even if employees are doing anything other then what they are supposed to do. These tools are expected to deliver greater accuracy without loosing performance. To achieve more accuracy in the analysis, the tools need to look into the content of the traffic, instead of just the transport data. Looking into the content of the traffic increases the resource need of the tool extensively and despite more and more powerful machines coming on the market, the amount of traffic to analyze increases as well.

A company in Akureyri, Iceland by the name of Stefna ehf. is working on a network traffic analyzing tool, Netvaki [1], that examines the material of traffic, and generates lists to filter out traffic coming from servers that serve illegal material. The first step of Netvaki's way of analyzing the network traffic, is to filter out non-interesting and non-important traffic according to the content of the traffic.

This project deals with estimating resource requirements for this filtering process and therefore a prototype of such filtering system was built.

This paper starts off by delivering some preliminary background material in chapter 2 including information about HTTP and the OSI reference model. The project is presented in chapter 3, where a description of the project is given, main requirements are cited and how they effect some decisions. Then in chapter 4 the design and implementation of the prototype is presented and thereafter, I describe the evaluation of the project along with the results from evaluation in chapter 5, and finally I will go over the key points together with a discussion on further research. Appendix A includes all the source code of the prototype.

[1] For more information on Netvaki or Stefna ehf. please visit http://www.stefna.is/ for contact information

Chapter 2

Preliminary background information

This chapter introduces preliminary material on several items that are discussed in this paper. Basics of network layers, the HTTP protocol and web caches are briefly described herein. Information about network sampling is given and the difference between content and transport traffic discussed. In the end a summary of what computing resources are is presented.

The OSI seven-layer model

In order to aid discussions of networks and their components we use the traditional OSI reference model. The Open Systems Interconnection Reference Model (OSI model) has seven layers, where each layer has a special function to carry out, which it does by some protocols. Following is a brief description of each layer in this model[10]:

- Layer 1: Physical layer: The physical layer defines all electrical and physical specifications for devices. Wires, fiber optic cables, hubs, modems, etc. are parts of this layer, and they communicate by sending bits or signals.
- Layer 2: Data link layer: The data link layer is responsible for transferring data (frames) between network items and to detect and correct errors from the physical layer. Here the Maximum Transfer Unit (MTU) is very important, and stands for the maximum byte size that is allowed enter this layer. The MTU plays a big part of this paper.
- Layer 3: Network layer: The network layer is responsible for transferring data items called datagrams or packets from a source to a destination. Addresses in this layer are chosen by the network administrator and are arranged in a hierarchical manner which allows for network routing. This layer also handles flow control, segmentation/desegmentation of packets from the transport layer, and error control.
- Layer 4: Transport layer: The transport layer deals with transporting segments from the upper layers to the network. This service is reliable and transparent.
- Layer 5: Session layer: The session layer handles the dialog between end-user application processes. Adjournment, termination and establishment of checkpoints are among the things that this layer does.

- Layer 6: Presentation layer: The presentation layer allows the application layer to forget about how data is represented. This includes MIME encoding, encryption, etc.
- Layer 7: Application layer: The application layer handles communication with the application process itself and delivers it onto the next layer.

Network traffic flows from layer 7 of the sender down to the 1. layer, over the physical link arriving at the 1. layer of the receiver, where the traffic flows up into the application layer. Figure 2.1 displays this movement of traffic.

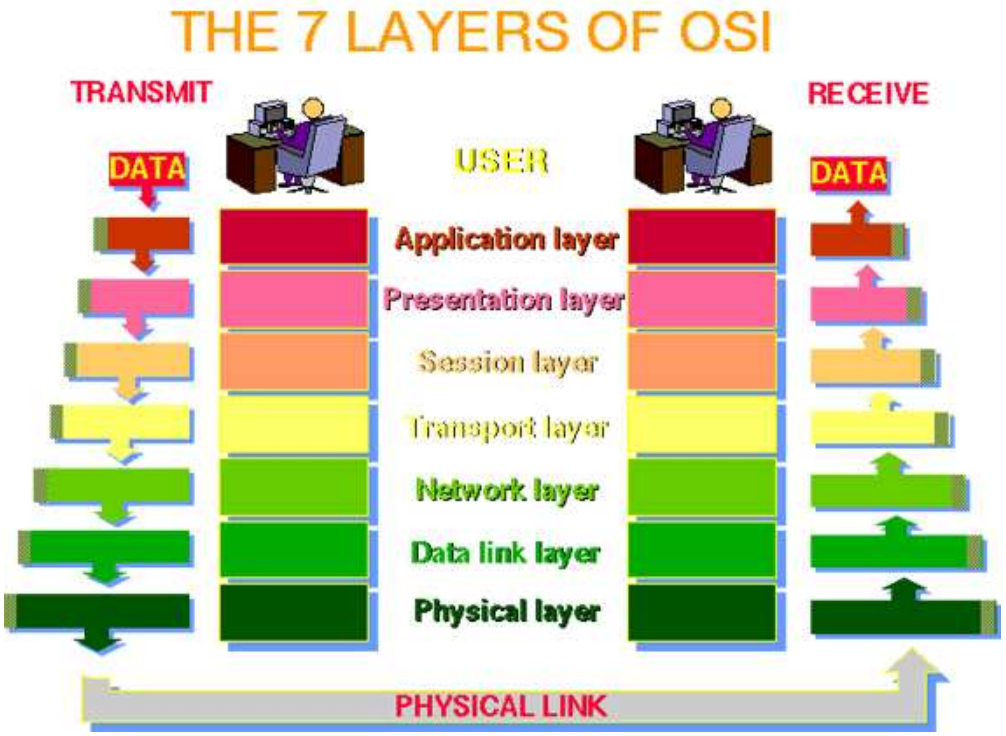

Figure 2.1: The flow of network traffic through the OSI 7-layer model[8]

HTTP

HyperText Transfer Protocol is the most used protocol of the Application layer protocols. It is the primary item in the World Wide Web. HTTP needs a client and a server. A client sends a request to the server who will respond in some manner. A Request can be of one of the following: GET, POST, DELETE, HEAD, TRACE and CONNECT. Each as a purpose but GET is the mot common one, and the only one that concerns this paper. A get request from a Firefox browser on a GNU/Linux machine looks like this:

```
GET / HTTP/1.1
Host: www.stefna.is
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.6) Gecko/20050223 Firefox/1.0.1
```

```
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

The responses have a numerical code, followed by an header and then the file. In this header there are several items that have a meaning to the client, like what kind of file it is, date, server information, etc. The corresponding response to the request above (stopping after encountering the beginning of the HTML document):

```
HTTP/1.1 200 OK
Date: Sat, 09 Apr 2005 13:10:54 GMT
Server: Apache/2.0.52 (Gentoo/Linux) PHP/4.3.9
X-Powered-By: PHP/4.3.9
Set-Cookie: PHPSESSID=a8e11039ba77219bf0bf3a092385fd49; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=ISO-8859-1

<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
...
```

A HTML document can include links to other files, that the browser will then fetch, the same way it did the HTML, by a GET request. In the HTTP header there is a mandatory item describing the type of file was fetched. This is the Content-Type item. The Content-Type as two parts, the general category it is in and the type itself. In the above sample, the file is a text document of HTML type. This is in fact the most common file type for text documents, and is referred to as the HTML document.

Proxy

A web cache or a Proxy, is a service that runs on a server and caches web pages and files attained from web servers. The client is configured to use one, and from there on, the client sends all requests to the proxy, who does is own request to the web server, get the response, and sends it back to the client. This way, the client never accesses the web server by itself. This is useful for protecting the client, and is often used also to filter out unwanted material. Note that by using a proxy, the server never knows exactly who is getting the information, and the client really does not know anything about the server it is getting the information from.

Network Sampling

Network sampling is the process of taking a sample of network traffic. This process is done at the physical layer, so the sampling device must be physically connected to the line being sampled. Sampling is primarily done in network debugging, analysis and criminal intent.

There are two things to consider while sampling: obtaining copies of the traffic, and filtering the traffic to obtain the relevant information. Following are the four most common ways to obtain the traffic:

- HUB: A hub is placed on the line being taped so that it can broadcast all traffic to the analysis machine.
- SPAN: Switched Port ANalysis is a special way of having a switch broadcast a copy of traffic to and from a port to another port
- TAP: Test Access Port is the preferred way of tapping a line. It is a designated device that replicates network traffic on a line and sends it to a special monitoring port
- Gateway: If a all traffic flows through a single server, called the gateway, this machine can run the analyzing tool.

Each technique has its pros and cons. HUBs are inexpensive but they can only handle half-duplex traffic and therefore can not handle extensive traffic loads. The SPAN method can be done easily on most manageable switches today, so setup time and planning is to a minimum. However, the switch can be flooded with traffic if the traffic load is up, since it is cloning the traffic. SPAN can also monitor multiple ports. TAPs have many advantages such as that if they go down, the traffic can still flow through, this is a very small device, and it can easily handle whatever traffic load it gets. The big disadvantage of this equipment is the price. Cost of TAPs range from the price of small server up to ten times that. Gateways servers can easily be used for obtaining traffic. They require no extra cost, since they are already there, but need to allocate resources for the analysis. Thus, might need more hardware, and are not able to perform their other function if they are there.

When filtering for data, the OSI model must be considered along with the idea of the two different types of traffic. Filtering can be done at any of the layers in the OSI model, but the outputs of the layers are different between the layers. Table 2.1 gives a brief description of what to filter for each layer.

Layer	Data unit	Purpose
Physical layer	signals	debugging of lines and devices
Data link layer	frames	debugging device drivers
Network layer	packets	debugging network stack of operating systems
Transport layer	segments	analysis of meta-traffic
Session layer and above	data	analysis of meta- and content-based traffic

Table 2.1: OSI layers and filtering

Content vs. Traffic data

There are two types of network traffic to consider: content-based traffic and descriptive traffic. Descriptive traffic is the network traffic that deals with making the all the different protocols work, and can be thought of as meta-traffic. Most of this type of traffic is based on the lower layers of the

OSI model, such as the network, transport and session layer, although that is not always the case. Examples of meta-traffic that reside in the upper layers are the DNS lookups, ARP lookups, HTTP requests, FTP control commands, etc. Content-based traffic is the collection of all network traffic that contains data regular users are after. This kind of traffic can only be in the upper layers, or the layers above the TCP/IP model, but the real problem is when the meta-traffic is in the same layer as the content. Using the same example as above: the HTTP data response containing the HTML and the images gotten from IMG HTML tags are all content-based. Things get even more uncertain when thinking about HTML itself. There the content is tangled with the HTML tags, which do not have a specific meaning to a user as content, and therefore should be considered as descriptive traffic.

Resources

Computers need resources to function. These resources can be memory, CPU cycles, disk space and more. The memory is needed to store information and programs that are being worked on, CPU cycles are used to perform actions, disk space is used to store static data that is currently not being worked on, and network access is needed to communicate with networks. The network load of the network is also very important, especially when performing analysis of network traffic.

Resource requirements of machines are dependent on the purpose of the machine. Some machines are built to focus on the users (the higher layers of the OSI model), while others are focusing on handling high network loads and perform actions on the lower levels of OSI model. A machine focusing on most or all the network layers will need more resources than others that focus on fewer layers.

CPU load in GNU/Linux is measured in jiffies, which is 1/100 ms, and is the way GNU/Linux is set out to measure loading information for processes and the whole system. Network load is normally measured in bits per second, or even bytes per second, usually in kilo- or mega-bits per second. Memory is measured in bytes, usually in MBs.

Chapter 3

The Project

This chapter gives the motivation for the project, details the requirements and itemizes many decisions made regarding the project.

3.1 Description of project

The Netvaki project deals with helping firewalls filtering out unwanted material by using methods of data mining and machine learning. Using methods like data mining and machine learning requires much computations, so to help improving the performance of these methods the data that is to be mined or learned must be stripped of everything that does not have effect on the outcome of these techniques. This means that packets such as normal TCP control packets can be discarded, because they do not contain any information about the material of interest. Packets containing sounds or any other material that is not content related text are also discarded because they most likely are only used for making the experience a little better.

While using HTTP as an example, the only document type that really contains information about the content of the web page, is the HTML document. The HTML document itself might be non-important because it might not contain the material in question.

Non-relevant traffic should be filtered out before the real analysis tools begin their work. The problem with filtering these things out, however, is that the packet must be dissected all the way up to the application layer (7. layer). This introduces many issues that are described in chapter 3.2.

To create meaningful estimates, a prototype filter was built. Other currently existing programs are not good enough for our purposes, as it is described in chapter 3.3.

3.1.1 Requirements of Netvaki

In addition to which layer to filter at, the Netvaki project put forward two requirements that effect the design of the prototype.

First of all, the placement of Netvaki will effect what can and will be filtered out. Netvaki will reside at the ISP level, where users cannot turn it off, or in any way get around it. By doing this, the total number of instances of the tool is lowered, and also allows for doing the analysis using a tool built to run on a multi-processor machine. Further, Netvaki is kept on the *Internet side* of

the ISP, meaning that it is the first unit a network packet goes through on its way to the user. This is in part due to privacy issues. By taking out the address of the user of the ISP, personal identification is impossible and many potential legal issues are avoided.

The second requirement has to do with performance. Analyzing network traffic for an ISP obviously needs quite a lot of power, and therefore, the Netvaki project will require a multi-processor machine to work on. Hence, for performance reasons, the prototype was designed to take full advantage of using more than one processor.

3.1.2 Prototype decisions

The purpose of building the prototype was to gaining some knowledge of the resource requirements for this kind of a program, several decisions were made regarding the magnitude of the project.

In this project, only HTTP traffic will be examined. Thus, all other upper layer protocols are discarded. HTTP is very common way of using the Internet, and is probably the most used way of getting information today. All exceptions from the normal network traffic will be discarded to minimize the time spent on the HTTP protocol itself.

The analysis will be confined to textual searches; meaning that only text is examined. The prototype will try to find IP addresses of servers that contain information about a given topic. The topic used will be concerning java and its API. Note, that it really does not matter what topic is chosen, it only has to do some searching.

3.2 Issues

There are plenty of issues involving upper layer network traffic analysis, specifically HTTP. Solutions to the issues raised here are considered in section 4.

3.2.1 Performance

Analyzing traffic at the ISP level requires that the software should at least be able to analyze traffic up to 1Gb/sec for a normal sized ISP. Even the biggest ones in Iceland are using that amount of traffic. Examination of Cisco PIX data sheets [4] suggests that a PIX that can handle traffic of 1.7Gb/sec is merely a 1GHz machine. The PIX however, only looks at headers of the traffic and does not deal with compression and segmentation is not an issue there because they do not examine the payload itself. Also, the PIX firewall is uses a specially built operating system that is built for this purpose only. That way they keep the hardware requirements down. By looking at these numbers, one can assume that analyzing the payload will cost more CPU power.

One way to solve the performance issue is to have separate processes doing separate jobs. This suggests a concurrent solution (as will be described in section 4), a solution in which a process that does desegmentation, while another is decompressing the document, while yet another one is searching full uncompressed documents for special keywords. This concurrent solution will be more effective than having one processor trying to do all of this sequentially for all packets.

Another way of solving this problem is to have multiple machines. Each machine could perform a task such as the separate processes before. This however has much more cost associated with it, and creates several problems with management.

3.2.2 Segmentation

Segmentation of network traffic is when a packet is split into segments in order to fit the maximum transfer unit (MTU) of a network. Normal Ethernet MTU is 1500 bytes. Most upper layer content related packets are much more than that, so most of this kind of traffic will be segmented. Since the analysis is done at upper layers, the analysis tool will most likely need to have the whole packet to analyze it, though some analysis can be performed on the segments themselves. The process of putting the whole packet together is called desegmentation.

In order to reconstruct the whole packet, all the segments of an incomplete packet must be kept in memory. This should not be a big problem for most HTTP HTML traffic, since normal HTML document rarely exceeds 2 MB. However, this could be a very big problem for protocols that deal with large files such as FTP. Focusing on HTML documents, another problem emerges, HTTP traffic is stateless; meaning that HTTP connections do not have to close, and therefore, it is difficult to determine the last segment of a packet. To accommodate this situation, the HTTP/1.0 specification [6] introduces a header item *Content-Length*. This item allows web servers to include the length of the responded message if the server knows that it will not close the TCP connection. Unfortunately, the Content-Length is not required by the HTTP/1.1 specification [5] even though the connection is not terminated.

3.2.3 Compression

Compression was a very crucial part of network traffic a few years ago. It still remains much used, but because of increasing speeds of networks, this feature is beginning to be less used. Compressing packets before they are sent usually does not take much time because normal packets are not so big. Protocols typically have an upper limit of the size that should be compressed, due to CPU power and memory that is needed to compress and decompress.

Many web servers, including Apache, have a option of compressing material before it is sent. The methods of compression allowed in HTTP are gzip, deflate and compress [6]. These are public/open methods of compression, and most browsers include support for them.

Decompressing an HTML document requires the whole document to be used, so decompression will take place after desegmentation. Despite the obvious disadvantage of this, there is a good side to it. Since web servers need to compress the document as a whole, the Content-Length of it is available.

3.2.4 Encryption

Today, privacy and other security measures require much of network traffic to be encrypted. This is of course a problem when analyzing the traffic. There is no way it would be computationally acceptable to try to decrypt all encrypted packets. Though it might be possible, it would just take too much time.

The way around the need to decrypt all packets in order to analyze the network traffic, we could look at the transport data instead, and perhaps create a profile of use, by, for example, calculating the percentage of traffic that is encrypted. However, even this can be difficult, since tunneling encrypts some or all traffic data. With IPSec, this is not possible, unless the transport data of the IPSec traffic itself is used.

3.2.5 IPv6

IP version 6 (IPv6), has a few problems and they exist in the security part of the protocol. In the future, some sort of IPSec will be built in it, which will introduce the same problems as mentioned in the encryption section. Today, IPv6 is not in general use and there are no intentions of usage in near future.

3.2.6 Privacy

For legal reasons, we need to ask: Are network analysis tools allowed to *look* at content related traffic? The quick answer is no. However, ISPs do it all the time, they scan your emails for viruses and SPAM. There is a general acceptance of using virus and SPAM scan engines to browse through content related email. The big question is: will the same acceptance be associated with overall network traffic?

3.2.7 Proxy

Proxy servers are a very popular for local area networks, especially in big corporate environments. Proxy servers are used to control the access of the employees as well as lessen the network load of the Internet connection. ISPs are using proxies for the same purpose. Due to the nature of proxies, some traffic data is altered and depending on which side of the proxy you are on, you are denied access to either the client or the server.

In a traffic analysis where addresses of both the client and the server is needed, having a proxy makes things more complex. There are different ways to implement a solution to this type of proxy problem. Building an extension for straight communication between the proxy and the analysis tool, is one way to go, while another solution include installing the monitoring tool on both sides of the proxy, and do the connection matching itself. Yet another solution is to have the analysis tool sit on the client side of the proxy, and dig into the HTTP proxy requests for URLs and host names, and use those to change the incoming packet accordingly.

If the analysis does not need either client or server addresses, the tool can be placed appropriately on the right side of the proxy, and discard the information that is not needed.

3.2.8 HTTP

Aside from the issues mentioned above, the HTTP protocol as described in 2 effects many possibilities for shortcuts in the analysis. First of all only responses with code 200 are basically important, along with the request itself. Secondly, unless you need to try to analyze image, videos, or even stylesheets, the content document[1] is the most important part of the analysis. If you are able to skip all except the content document then most of the traffic load can be dropped.

[1]Here the content document is though of as text/html, text/plain, and text/xml. All the text files that contain data

3.3 Related work

There are quite a few network analysis tools being used today and some are better than others. After considerable amount of time spent on searching for good tools to do the job, I found that the most likely ones would be Snort, tcpdump and Ethereal. These tools are open source, so they could be modified if needed. However, these tools do not take any steps towards utilizing multi-processing, and filtering at the upper layers needs to be specially built. Therefore, these tools cannot help this project in any other way, than just giving ideas of how to do certain specific things like decompressing HTML documents and general TCP connection tracking.

This project is unique because it specializes in using multi-processing machines to increase performance, thus creating different results from the rest of the tools. According to my study no one has tried to build estimates that deal with upper layer network traffic analysis.

Chapter 4

Design & implementation

Herein the design and implementation of the prototype is described. The techniques for obtaining network traffic are enumerated, and the architecture and code implementation of the prototype is described in detail.

4.1 Sampling the traffic

The Netvaki project team does not have a preferred placement for the tool itself. Therefore, any method of getting the traffic as described in section 2 can be used. I choose to use a Gateway. (see section5.1 for details on why this particular method was selected). By using this method, no extra hardware or software is required and there is no need to change anything in the environment.

4.2 Solutions

In section 3.2 the issues related to network traffic analysis at the upper network layers were presented. In this section each of these issues is discussed separately, and designed solutions are presented.

Performance

The performance issue really influences the architecture of the prototype software. The requirement for multi-processing is met by using threaded pipeline of filters. Each thread will be able to be run on separate processor if possible, and work on its part of the filtering process. Using a pipeline helps distinguish between the tasks, and delivers a simple but powerful way of exchanging data. The pipeline was chosen because each thread has to finish its job, before the next one continues filtering (if needed).

Segmentation

The process of desegmentation in the prototype is as follows: If a segment contains an HTTP header but not Content-Length, then that segment will be regarded as the only segment in the packet, and therefore, the whole packet. Note that there is no attempt to look for closing connections. The

reason for that is that if a FIN packet is expected, and it does not arrive, then there is perhaps a large portion of memory that will never be used again. The way around that problem can be implemented by using periodic checking for packets that have been there for too long.

If the segment contains an HTTP header and the Content-Length, the following segments will be appended to the previous segment, until that segment has reached a similar size as the content-length value designated. Note, that after quite a bit of HTTP traffic analysis using Ethereal, I noticed that some packets did not have the exact length, so there is a leeway for packets to be 10 bytes more or less that the content-length proposed.

If a segment does not contain header data, then it will try to find a match in the segmentation table described in section 4.3, and if there is no match the segment is deleted. Note, that this leads to some traffic data not being analyzed. The reason for deleting instead of trying to analyze, is that the segment might lead to misjudgment because the context is not right. Also, unpacking cannot be done on incomplete data.

Compression

The check for compression, just checks the HTTP header for Content-Encoding to equal compress, deflate or gzip. From there the data is uncompressed into a big buffer, and from there the newly uncompressed data is used to generate a new packet containing all the headers from the compressed packet, and then the compressed packet is deleted.

If there are any errors while trying to decompress, the packet is deleted. There is no reason for trying analyze compressed data in the same way as uncompressed because compressed data is in binary form, and cannot be analyzed textually without decompress it first.

Encryption

This prototype does not solve the encryption issue. Since its general goal is to perform textual analysis of the HTML documents, there is no need for the transport traffic data that follows the encryption, and as said in section 3.2.4 it is computationally too hard to decrypt packets.

Privacy

By following the Netvaki requirements and architecture, there is no need to address this issue.

Proxy

By following the Netvaki requirements and architecture, there is no need to address this issue.

HTTP

The prototype will only focus on HTTP response 200 and text files of type HTML. Plain text, XML files and other content documents could be a part of this, but since the analysis of those files is identical to the HTML one, and that these files are not as common as the HTML files, they are not used.

4.3 Architecture

The architecture of the program is a multi-threaded pipe-lined filter. Each filter is a thread that gets a input and delivers output. The pipe itself is created from scratch using a thread-safe first in first out queue. Following are the main parts and the classes of the program as well as descriptions of each part. Figure 4.1 shows the relationship between the threads of the prototype.

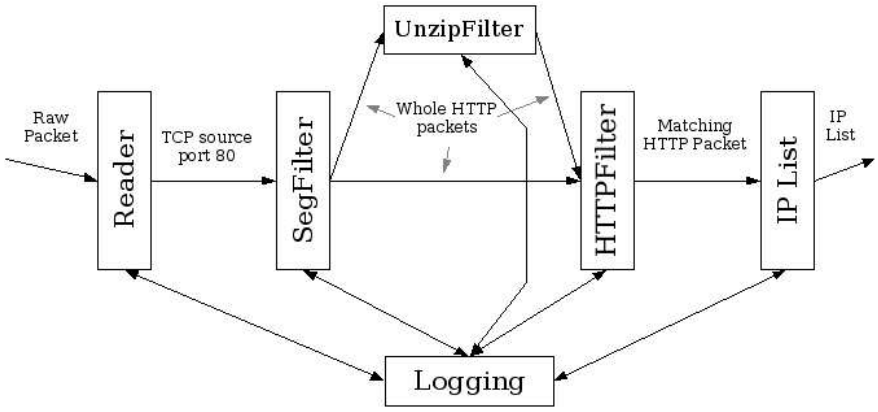

Figure 4.1: The overall architecture of the prototype

Main

The main thread serves as the coordinator. The main thread starts everything up, initializes all threads and variables, and fetches counting data from the other threads. The main thread also contains code to handle all signals, including the reconfigure signal (SIGHUP) and termination signals. When a termination signal is found, the thread tries to stop all the other threads, by calling their stop method.

Once all threads are started, the main thread fetches the newest counts from the threads every 1 second. Method calls from the main thread get the counts, that are then outputted to the screen[1].

The counters include: number of packets coming in, total size of the packets, user and system CPU usage in jiffies, current number of items along with maximum number of items for the segmentation table, and also the number of items in the last filter's answer table.

NetworkReader

The NetworkReader is the thread that does the actual traffic sampling by getting the packets of the network interface, and creates Packets. The reader discards all traffic not coming from TCP port 80, i.e. HTTP responses. Furthermore, it checks to see if the payload of the packet is more than 10, because 10 is the number of bytes that can be used as a leeway in the desegmentation

[1] uses standard out, so it can be piped or filtered at will

process. This removes all control messages, such as the TCP three-way handshake, and others that do not contain any content.

Packet

The Packet is created each time a valid packet destined for Filter2 from the NetworkReader. The packet is a class that contains a string of bytes, i.e. the packet itself, also has pointers to the Ethernet, IP and TCP headers with the possibility of referencing those with the right semantics. Meaning that to get the TCP source port, we do `packet.tcp.src_port` and it automatically find the reference to TCP in the packet, and find the source port from there[2]. The packet also contains sizes of all headers in the packet along with the payload size.

There are two constructors for Packet, one for the NetworkReader, where only the string of bytes from the network interface is copied, and the other for the Segmentation table to recreate on Packet from two others. Other than the constructors and the destructor, Packet does not have any functions, all done to minimize the extra memory needed to create a Packet. Remember, Packet is create every time the reader reads a *valid* packet.

Filter

The Filter class is the super-class of all the Filter classes (segFilter, HTTPFilter and IPList filter). This class contains methods that are common to all of them, such as the interaction methods that are called from the Main thread for counters. Semaphores are used to ensure mutual exclusion of the counters.

segFilter

The segmentation filter thread gets the Packet from the reader, and tries to create a Header object from the Packet. If the Packet does not contain a HTTP header, the header return an error, else it creates a Header object that will be associated with the Packet by creating a HTTPPacket. From the Header object, the response code and the file type is read, and HTTPackets with the wrong code or file type are discarded. The packets that get through carry on and next the Content-Length is read, and if it is valid, and needs more bytes, the HTTPPacket is put in the segmentation table. HTTPPackets that have the whole content are sent directly to Filter3.

Packets not containing an header, are sent to the segmentation table in order to search for parents. If no parents are found, the Packet is deleted. (See following section for detail on the segmentation table). If a parent was found and the HTTPPacket has all its bytes, it is sent to Filter3.

Segmentation table

The SegTable object resides inside the Filter2 object. It contains a static table of 1000 items of type segitem. Segitems are a struct with HTTPPacket, length of that packet, and a flag whether the item is free or not. A static table was chose to lessen the load of the machine due to memory allocation.

[2]Done by using a C **struct**

When the first segment is sent to the table, it finds a free item, and the HTTPPacket is placed there, and the length of the item is set to the length of the total payload size. For each segment that follows the table is searched for a prior segment that has the same TCP destination port, TCP ack number and IP source address. If found, a new Packet is created from the parent and the new packet. The reference to the Packet in the HTTPPacket object is then updated to the new Packet. When the full length of a HTTPPacket is found, it is removed by only setting the segitem free flag to true and that packet is returned.

Header

The header class is used to get and maintain the HTTP header of a packet. Its constructor takes in a reference to the Packet, and checks if the packet is a header by checking if the first 4 bytes are "HTTP". If that fails, no other actions are taken and the error flag is set for the header. However, if it works, then the HTTP header items are read sequentially into a static array of headers.

Methods such as getResponse and getValue search the array for matching item, and return the corresponding value.

HTTPFilter

This filter gets HTTPPackets from the segFilter. First task is to check if the encoding matches some of the compression types. If the answer is yes, then an attempt to unzip the packet is made. (The unzipping process is discussed in section 4.4).

The next task is the textual analysis. Regular expressions are used for the search. The regular expressions are read in through the configuration file and then compiled. There can be up to 15 regular expressions, all with the maximum length of 100. These regular expressions are associated with points that are given when a pattern is found. By using this method, one can both expand and narrow searches, and use this to control the searching in any way.

If a document is scored at 70 points or more, an HTTPItem is created and sent to Filter4. The documents that do not reach the threshold are deleted.

HTTPItem

HTTPItem is a small class that contains the packet number, points given, IP source address and the size of the Packet. Note, that it does not contain any reference to any Packet or HTTPPacket.

IPList filter

This is the last filter thread. This filter maintains the IP table of the matched sources. HTTPItems coming from Filter3 are put into an array of HTTPItem only if the total points of an item does not exceed a certain threshold of 500 points. This is done to minimize the memory usage of the program.

BQueue

The BQueue class is used as the thread-safe FIFO queue between the filters. It uses a producer consumer in a bounded buffer paradigm. Use of signal and wait ensures mutual exclusion on the

data buffer. The buffer can contain 16384 items, and each item is a pure pointer to an object using Cs void pointer and casting it to the right type after the pointer has been fetched from the queue.

4.4 Implementation

This section contains a description of the prototype implementation, comments on particular languages and libraries used in the implementation are given.

- Language: The prototype was implemented in C and C++, mainly for performance reasons. Although most code is in C, C++ classes are used to utilize the constructor and destructor features for objects instantiated therefrom. I chose C and C++ for programming languages, since they are known to produce very fast code. I mainly use C style of coding, except for I am using C++ classes, because of the convenience of constructors and destructors and also because I have experience in that language.

- Operating System: I choose GNU/Linux for the operating system to be used, because devices are presented in an uniform way of block and character devices. This uniform way allows a program to listen on **ANY** network interface, and therefore, allows the programmer to listen to more than one device at a time. Further, GNU/Linux is a very stable and secure operating system that makes the programmer's life a little easier by including source, header and instruction files where they need them.

- Libraries:

  - pthread: Since Linux was chosen, the pthread library is the best step for threading. Linux maintains the threads in the kernel, and therefore, can schedule the threads onto multiple processors easily and effectively. Note, in Linux kernel headers 2.6 these threads are not split onto separate CPUs, but rather processed as one process. To use the pthread effectively on a 2.6 kernel, the following command is needed before running the program: `export LD_ASSUME_KERNEL=2.4.1`.

    I also found a very interesting class POSIXThread[22] that creates a virtual thread class that uses the pthread library. This allows for creating an Object Oriented thread class.

    Inside the pthread library, are also routines that are called conditions, where the programmer can tell a thread to wait until it get a signal. The signal can then be signaled to the next thread waiting, or even a broadcast to all. This is used in the BQueue class to ensure that the buffer is not accessed by two threads at the same time.
  - libpcap: PCAP library[23] is the most used packet capturing library in the UNIX environment. This hides all the issues with actually sniffing a network interface, and allows the programmer to simply capture packets.
  - zlib: The zlib library includes functionality to compress and decompress data in deflate and gzip format all in the same inflate function. The actual unzipping method is derived from the tvb_uncompress of the tvbuff.c file in the epan directory of the Ethereal source code[24].

- regex: I use the POSIX regular expression library for regular expressions (RE). This is the most common RE library used in GNU/Linux. They are really easy to use, and for programmers that have to deal with REs this is a must have library.

- Process and machine loads: In GNU/Linux every process has a directory in the **/proc** directory containing a file **stat** that holds the number of jiffies that the process has had the CPU for, both user and system time. There is also a file **/proc/stat** that holds information about the total amount of jiffies the CPUs have performed for, along with the total number for all processors.

  I found the best way of getting this information was to keep the process ID for all threads and read from the corresponding */proc/¡pid¿/stat* file and the total CPU file. This is done each time the Main thread gathers the counters and prints out.

- Compiler: I use the standard GNU C++ compiler, g++, which is a part of the known gcc compiler. In GNU/Linux, using this compiler is the easiest and most common. I also use their debugger gdb for most of the debugging, along with the ddd[25] helper tool.

- Coding: All source code is edited with either vim or gvim on a GNU/Linux system.

Chapter 5

Experimentation

To evaluate the prototype and to be able to derive some conclusion about the estimation, some testing was performed. This chapter discusses how the experiment was designed and how the results are to be interpreted.

5.1 The method

This section describes the methods used to test and evaluate the prototype. Included are methods to build statistics of the program's CPU usage, and observations on how network traffic loads effect CPU usage.

The test site was the computer science computer lab of University of Akureyri (R316). It has around 30 Linux client machines and all traffic flows through one server (viti). Viti has 4 Intel XEON processors of 2.2GHz each, all supporting hyper-threading. Thus, the Linux operating system treats the server as if it has 8 CPUs when running a threaded program.

One of the biggest problems encountered in this project was the problem of generating network traffic for testing purposes. The traffic must at least include HTTP packets, so the program can do some work. It should be possible to generate traffic up to 80Mb/sec, so that it is easier to see what happens at extreme loads. I created a small shell script that utilizes the wget utility. Wget has options of recursively crawling through web sites, following all links they encounter. By using wget I can manually pick a website, and make wget generate the traffic itself, also by using wget, no hits will be detected as cached. The script spawns 6 wget processes for separate web sites.

I used a remote connection to the client computers of the lab to run the robot crawler. At most I was using 14 computers at one time. The traffic load was controlled by the number of clients running the robot. I had problems with generating a steady stream of packets coming in. There were gaps in the flow, which lead to non-normal distribution that makes deriving statistics much harder. These outliers are usually at the upper boundaries of the network load, so to fix this I had to generate more load, and just simply cut out the outliers afterwards.

Besides testing the network load versus the processor load, I took a look at how different regular expressions had an effect on the CPU load. I tested with 1, 5 and 10 regular expressions. I structured the REs as such that they narrow the search down with each RE. This can give results that show how much output can decrease with each pattern, and maybe find a limit of how many

REs one should use, before CPU load starts to get effected.

All the output from the prototype, i.e. all the information about the network load and the cpu load is saved into a file, and then later processed using statistical analysis tool such as Open Office Spreadsheet and SPSS. The statistics are then used to generate predictions about the state of a machine when under certain network traffic load. This statistic can also be used to show what part of the process is most demanding, and what is the least demanding.

The execution of the experiment had 6 parts; for all three pattern numbers 1,5 and 10, the experiment ran with both 2 client and 7. The single pattern was simply the word java. Thus, it performs a basic word search. The 5 patterns were aimed at trying to find java APIs by including several words contained in the official API for each of the 5 patterns. The last one was designed to bring more accuracy to the 5 pattern search, by holding more patterns but an equal number of items in the pattern. There was also a test run with 14 clients and 1 pattern, to help seeing how accurate the predictions are. Each test ran for 2 minutes, producing 120 cases of data. The filter began execution after the robots had been started and the filter was stopped before the robots were stopped.

5.2 Results

This section describes and discusses the results from the testing. I will go through the statistics that were derived, and make suggestions on what they mean.

5.2.1 Data

By exploring correlations of individual variables of the data, only a few variables were decided to be significant, and the rest of the variables were discarded.

Following is a list of the significant variables. Note that the number of jiffies consumed be a thread in a second is a measure of the how much of a CPUs time the thread takes. Since one processor executes 100 jiffies in a second, the number of jiffies consumed are also the percentage of usage for a single processor. The consumed jiffies represent the load of a thread.

- Load of reader thread.
- Load of segmentation thread.
- Load of the search thread.
- Total load of the program.
- Network traffic load, in kilobits per second.

Variables such as the load of the unzipping thread and the IP list thread not significant because they did not consume any considerable amount of jiffies. The main thread however is discarded here because its load was generated when creating this data.

From hereon, the significant data is referred to as data and the discarded data is not considered further.

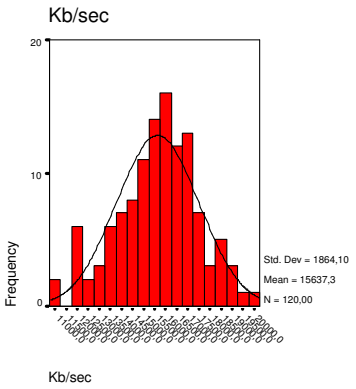

Figure 5.1: Distribution of traffic load. 1 pattern and 7 clients used.

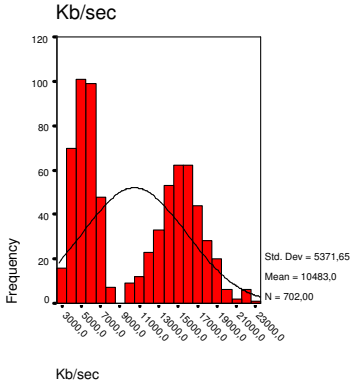

Figure 5.2: Distribution of traffic load. All pattern and clients used.

For each of the 6 individual tests, all the data had a reasonably normal distribution as shown in an example histogram in figure 5.1. However, when all tests are considered the distribution suffers from lack of continuity (figure 5.2), which leads to less descriptive and accurate information.

5.2.2 Correlation

The correlation between the variables is best described in scatter graphs and regression lines.

1 Pattern Search

In figure 5.3 the relation between traffic load and the load of the segmentation thread is shown. R^2 is 0.5931, so there is some relation between the two. This is natural since, there are usually some portion of network traffic that is segmented, and therefore, the segmentation thread needs to work accordingly. As shown in figure 5.4 the reader is also influenced by the traffic load, since the reader is the first thread to encounter the the traffic. The search filter is not showing much traffic in figure 5.5 because there is only one pattern to search for, and the thread seems to handle it easily.

The load balancing of the three major filter is shown in a pie chart in figure 5.6. The search filter does not take up any considerable amount of time, while most of the work is performed by the reader thread.

5 and 10 Pattern Searches

When more patterns are used by the search filter, the balancing of load starts to change. The pie chart in figure 5.7 shows how much the search thread has taken over when it needs to search for 10 patterns.

The difference between the patterns is best described in figure 5.8. There is a very clear distinction between the regression lines of each number of patterns group. This filtering effects the overall program load significantly as shown in figure 5.9.

5.3 Conclusion

By extending the regression on the previous covered statistics, the future values of the data can be found according the regression equation. Figures 5.10, 5.11 and 5.12 show the predicted values the reader, segmentation and search threads respectively.

Reader thread

According to the prediction figures, the CPU load of the reader thread will hit the 100% mark at around 150Mb/sec. Many ISPs in Iceland only have a 100Mb/sec link. According to [26] the biggest service providers, Siminn, Lina.net and Islandsimi are able to generate traffic over 150Mb/sec. The Netvaki project is aiming for the biggest ISP in Iceland to begin with, and perhaps later in Europe. These results are bad news for the project, but suggestions on how to improve the performance of the reader are discussed in the following section.

Segmentation thread

The segmentation thread manage a little more network traffic load then the reader, up to 180Mb/sec. According to [26], both Siminn and Lina.net are above that number, and others are near. Again, this is not good for the Netvaki project, although improvements can easily be made to this thread (see next section).

search thread

The search thread however can only handle up to 50Mb/sec using 10 patterns, and 80Mb/sec using 5 patterns. Many ISP and companies are reaching these numbers in their network load, and it is clear that something radical must be done to this thread in order to be able withstand loads of big ISPs. On the other hand using the single pattern, single word search the search thread reaches over the 1000Mb/sec limit, and this is a good sign for Netvaki, who will likely use this to their advantage.

overall

The overall load is related to the number of patterns used, and with more than 1 pattern the overall process will hang before reaching the limit of e.g. Lina.net of approximately 300Mb/sec. Even when considering only one pattern searches (or even no searches), this type of filtering will not work for Lina.net. Major improvements are needed on the whole process, especially if a large European ISP is considered.

Correctness

To see if the statistical analysis was correct, I performed a test of a single pattern using 14 client, generating up to 38Mb/sec. Comparing the actual results from that test with the predicted values, demonstrated that the experiment was a success and the predictions can be trusted.

5.4 Reflection and improvements

Herein, are several suggestions on how the performance of the prototype can be improved. None of the suggestions should involve major change in the program. Further improvements and development is described in the discussion chapter.

Increasing the performance of the reader can be done by perhaps using multiple readers. Having multiple readers however requires multiple interfaces, or some load balancing between the readers. Reading from multiple interfaces has to do with splitting the traffic at either layer 2 or 3 while using a switch. An extra machine between the network sniffing point and the filtering tool, can also split the traffic according to some routing rules.

Another way to enhance the readers performance is to rewrite or skip the pcap library. The library has several function that are not needed, and therefore, can be skipped. Writing a very simple and fixed solution might allow for significant increase of traffic load being read before CPU load hits critical levels.

The segmentation thread is only using a simple array, and will go through the whole array if needed to find a match. Using a tree structure should be very easy to implement, and will bring the segmentation search to $O(\log n)$ from $O(n)$. There is also a choice of having more than one segmentation thread, but still use the same data structure.

Improving the search thread is not needed when using the single word pattern, and by creating a pipeline of such filters, the search can be executed on more than one CPU. By creating several copies of the current search thread, and load balance between them, the searches can include more patterns and they can be more complex. There are also other methods of searching than can improve both the performance and accuracy of the search.

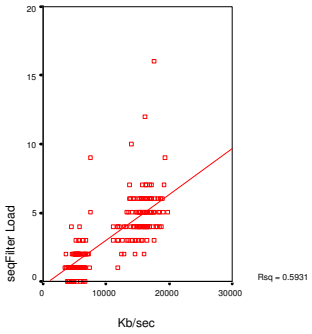

Figure 5.3: Relationship between segFilter load and network traffic load. 1 pattern used

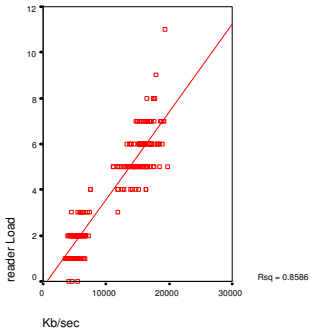

Figure 5.4: Relationship between reader load and network traffic load. 1 pattern used

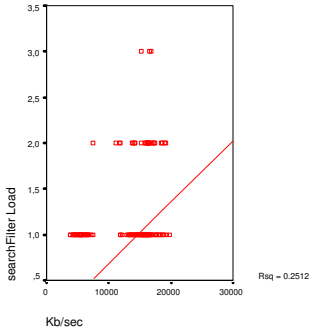

Figure 5.5: Relationship between search thread load and network traffic load. 1 pattern used

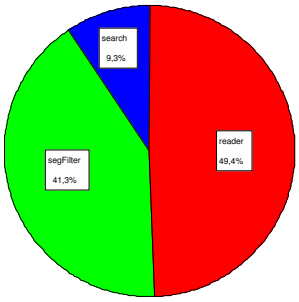

Figure 5.6: Distribution of load using 1 pattern

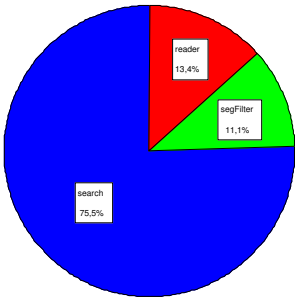

Figure 5.7: Distribution of load using 10 patterns

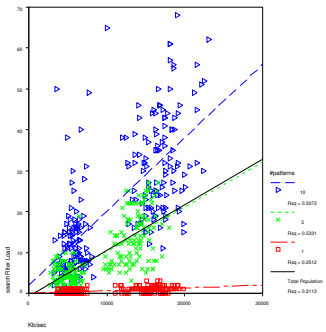

Figure 5.8: Relationship between search thread load and network traffic load

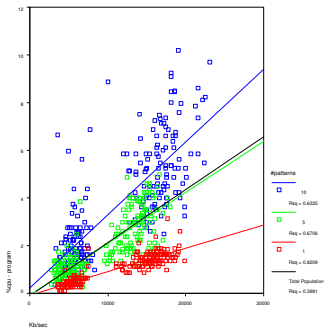

Figure 5.9: Relationship between program load and network traffic load.

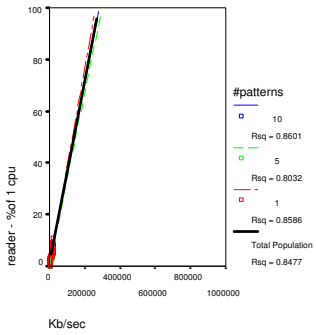

Figure 5.10: Prediction for the load of the reader thread according to traffic load

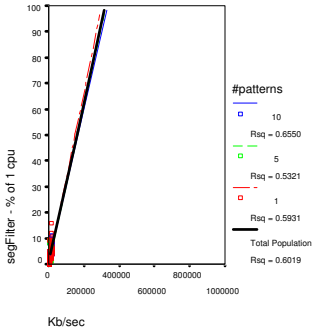

Figure 5.11: Prediction for the load of the segmentation thread according to traffic load

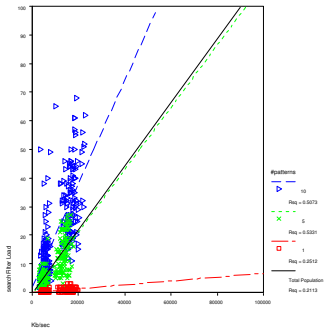

Figure 5.12: Prediction for the load of the search thread according to traffic load

Chapter 6

Discussion

The project described herein, was designed to help the Netvaki project team estimating needed resources for network traffic analysis that analysis at layers above the transport layer. The project also makes the first steps of building the primary sniffer and filter for Netvaki easier by creating a prototype of such tool. Further, the project discusses ideas for improving the prototype for increased performance.

Aside from helping the Netvaki project, this project contributes to interested parties resource estimation along with bringing out issues with network traffic analysis at the higher layers and how they can be solved in a multi-processor environment. Research on this specific topic has not been available, and now that machines are becoming more powerful, companies will begin creating devices that work on the higher levels. This paper should benefit those companies.

As with all things, this is not a complete study. This is merely the beginning. Further studies include improving the efficiency of this project's prototype described in section 5.4, altering the prototype to allow for the filter to reside on different machines and thus, creating a distributive analyzing tool, and since this project decided to discard issues like proxy, missing content-length and file types other than text/html, solutions for these issues is need for the general analyzing tool.

Even further research on the data and its statistics can be performed without making extensive changes to the prototype. Creating more conclusive and extensive data sets to work out statistics and estimates with more accuracy is very important, as well as extending the experiment to include data on IO and memory usage. To gain a better estimation of the actual computing power needed, the experiment can be performed using different types of machines, single processor, more than 8 processor machines, slower and faster machines, etc.

There are plenty of possibilities for studies in this field, and hopefully there will be many more. At least I will continue to develop and improve this prototype even further along with helping the Netvaki team create their solution.

I would like to say that this project was a partial success. A prototype was created and used to create estimates of resources. However, the time it took to design and build the prototype exceeded all time-plan by a big margin. I would have liked to use the time for more data building, and perhaps some of the improvements described in section 5.4. Nevertheless, many issues were found and some solved, some ideas of the resources needed was proposed and ways to improve the performance of the prototype were introduced. So, all in all, the project was a success.

Bibliography

[1] A. Escudero-Pascual, I. Hosein, *Questioning Lawful Access to Traffic Data*, ACM, 2004.

[2] Andrews, Gregory R. 2000. *Multithreaded, Parallel, and Distributed Programming*. United States of America. Addison-Wesley.

[3] Moore, David S. and McCape, Gregory P. 2003. *Introduction to the Practice of Statistics*. New York. W. H. Freeman and Company.

[4] http://www.cisco.com/go/pix,
last accessed April 15, 2005

[5] http://www.ietf.org/rfc/rfc2616.txt,
last accessed April 15, 2005

[6] http://www.ietf.org/rfc/rfc1945.txt,
last accessed April 15, 2005

[7] http://www.erg.abdn.ac.uk/users/gorry/course/intro-pages/osi-example.html,
last accessed April 15, 2005

[8] www.webopedia.com/ quick_ref/OSI_Layers.asp,
last accessed April 15, 2005

[9] http://www.usenix.org/publications/library/proceedings/bsdcon02/
full_papers/lemon/lemon_html/node2.html,
last accessed April 15, 2005

[10] http://en.wikipedia.org,
last accessed April 15, 2005

[11] http://markun.cs.shinshu-u.ac.jp/learn/osi/,
last accessed April 15, 2005

[12] http://freespace.virgin.net/glynn.etherington/data_encapsulation.htm,
last accessed April 15, 2005

[13] http://www.wilsonmar.com/1isotp.htm,
last accessed April 15, 2005

[14] http://www.rabbitsemiconductor.com/documentation/docs/manuals/TCPIP/Introduction,
last accessed April 15, 2005

[15] http://www.ciscopress.com/articles/article.asp?p=98156&seqNum=2,
last accessed April 15, 2005

[16] http://navigators.com/internet_architecture.html,
last accessed April 15, 2005

[17] http://www.cs.columbia.edu/ hgs/internet/,
last accessed April 15, 2005

[18] http://www.helsinki.fi/ ksiazkie/nerd/internet/infrastructure.html,
last accessed April 15, 2005

[19] http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ip.htm,
last accessed April 15, 2005

[20] http://www.tcpipguide.com/free/,
last accessed April 15, 2005

[21] http://telecom.tbi.net/how-dial.htm,
last accessed April 15, 2005

[22] http://www.partow.net/programming/POSIXsynchwrapper/index.html,
last accessed April 15, 2005

[23] http://www.tcpdump.org/,
last accessed April 15, 2005

[24] http://www.ethereal.com,
last accessed April 15, 2005

[25] http://www.gnu.org/software/ddd/,
last accessed April 15, 2005

[26] http://www.rix.is/statistics.html,
last accessed April 15, 2005

Appendix A

Appendix A includes the whole source code of the prototype. Files are displayed in alphabetical order, and are separated with the header of the next file.

```
/*
   Project:       Final year project in CS in UNAK
   Author:        biggistefna.is

   File:       biggisfilter.cpp
   Summary:    Main methods for the project
       along with coordinator and logging thread.
*/

#include "funcs.h"            // get_hwaddr, debug
#include "networkreader.h"          // the network reader
#include "bqueue.h"           // queues
#include "segfilter.h"            // segmentation filter
#include "search.h"           // search filter
#include "filtz.h"            // filter zip
#include "iplist.h"           // ip list filter

#include <stdio.h>
#include <iostream>
#include <pthread.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <time.h>
#include <fcntl.h>
#include <sys/resource.h>
#include <stdarg.h>


FILE *fdebug;
char debugfile[] = "main.debug";

BQueue q1("queue1"),q2("queue2"),q3("queue3"),qZ("queueZ");

NetworkReader reader(&q1);

IPList iplist(&q3);
Search search(&q2,&q3);
FiltZ filtz(&qZ,&q2);
SegFilter segfilter(&q1,&q2,&qZ);


int mainpid;
char _configFile[100];
char *configFile;

// array of pids
#define NUM_PS 6 //main+reader+filt*4
int pids[NUM_PS];
#define PID_MAIN 0
#define PID_READER 1
#define PID_FILTZ 2
#define PID_FILT2 3
#define PID_FILT3 4
#define PID_FILT4 5

void debug(int d, char *name, char *str, ...)
{
#if DEBUG_B >= d
```

```
   if (DEBUG_B>=d)
   {
      va_list argp;
      va_start(argp,str);
      fprintf(fdebug,"DEBUG(%s(%d)): ",name,d);
      vfprintf(fdebug,str,argp);
      fprintf(fdebug,"\n");
      fflush(fdebug);
      va_end(argp);
   }
#endif
}
void stopAll()
{
   reader.stop();
   debug(1,"main","Now, asking filters to stop");
   //filt1.stop();
   segfilter.stop();
   search.stop();
   filtz.stop();
   iplist.stop();
   debug(1,"main","Filters have stopped, now for logger");


   debug(1,"main","waiting for everybody to finish");

   reader.join();
   debug(2,"main","joined the reader");

   //filt1.join();
   segfilter.join();
   search.join();
   filtz.join();
   iplist.join();
   debug(2,"main","joined the filters");

   debug(2,"main","joined the logger");

   exit(0);
};
void terminate(int x)
{
   int signalpid = getpid();
   debug(2,"main","Got a signal(%i). Stopping(pid=%d)\n",x,signalpid);

   if (signalpid==mainpid)
   {
      stopAll();
   }
   else
   {
      debug(2,"main","hmm... this is not the main thread mainpid=%i - mypid=%i\n",mainpid,signalpid);
   }
}
void newFailed()
{
   fprintf(stderr,"A function calling new has failed, Failed to allocate storage\n");
   stopAll();
};
void reconfigure(int x)
{
   if (getpid()!=mainpid)
      return;
   FILE* f = fopen(configFile,"r");
   if (f==NULL)
   {
      fprintf(stderr,"Could not open %s for reading",configFile);
      exit(1);
   }
   search.flushPatterns();
   char line_t[150];
   char *line = line_t;
   char *key;
   char *value;
   u_int keylen = 0;
   u_int strlength = 0;
   u_int vallen = 0;
   while (fgets(line,150,f)!=NULL)
   {
      debug(2,"main","Got line %s",line);
      while(isspace(*line))
         *line++;
      if (*line == '#' || *line =='\n' || *line == '\0')
```

```
            continue;
        debug(2,"main","komst hinga (%i)",*line);
        strlength = strlen(line);
        key = line;
        value = index(line,' ');
        vallen = strlen(value);
        keylen = strlength - vallen;
        while (isspace(*value))
            *value++;
        if (strlen(value)<1)
            continue;
        debug(2,"main","Value = %s",value);
        if (strncasecmp(key,"pattern",keylen)==0)
        {
            search.addPattern(value);
        }
        else if (strncasecmp(key,"maxpackets",keylen)==0)
        {
            int c = atoi(value);
            reader.setPacketsCount(c);
        }
        else if (strncasecmp(key,"filter",keylen)==0)
        {
            reader.addFilter(value);
        }
    }
    search.compileREs();
    fclose(f);
}
void usage(char *s)
{
    fprintf(stderr,"Usage: %s [-d dev] [-c configFile]\n\n",s);
    exit(1);
}

void readPIDStat(int pid, unsigned long long *user, unsigned long long *sys)
{
    char filename[80];
    char s[1024];
    char *S = s;
    int fd;
    int num_read;

    sprintf(filename,"/proc/%d/stat",pid);
    fd = open(filename, O_RDONLY, 0);
    if (fd==-1)
    {
        fprintf(stderr,"Could not open %s\n",filename);
        *user = *sys = 0;
        terminate(19);
        return;
    }
    num_read = read(fd,S,1023);
    close(fd);
    if (num_read<=0)
    {
        fprintf(stderr,"Could not read from %s\n",filename);
        *user = *sys = 0;
        terminate(19);
        return;
    }

    S[num_read] = '\0';
    S = strchr(S, ')') + 1;
    *S =+ 2;                            // skip ") "

    unsigned long long utime,stime,cutime,cstime;
    int i;
    for (i=0;i<11;i++)
        S = strchr(S,' ') +1;


    //printf("S=%s",S);
    int num,n;
    num = sscanf(S,"%Lu %Lu %Lu %Lu %n", &utime, &stime, &cutime, &cstime,&n);
        //printf("%d - %Lu %Lu %Lu %Lu\n",num,utime,stime,cutime,cstime);

        *user = utime;
        *sys = stime;
};
void readCmdLine(int argc, char *argv[])
{
    for (int i=1;i<argc;i++)
```

```
    {
        printf("Checking %s\n",argv[i]);
        if (argv[i][0]=='-')
        {
            switch(argv[i][1])
            {
                case 'd': //device
                    i++;
                    if (i==argc) usage(argv[0]);
                    reader.setDev(argv[i]);
                    break;
                case 'D': //debug
                    break; // not implemented
                case 'c': //configfile
                    i++;
                    if (i==argc) usage(argv[0]);
                    configFile = argv[i];
                    break;
                default:
                    usage(argv[0]);
            }
        }
        else
        {
            usage(argv[0]);
        }
    }
}
void segfault(int x)
{
    fprintf(stderr,"Got SIGSEGV. My pid=%i\n",getpid());
    sleep(2);
    exit(100);
};
void getCPUUsage(unsigned long long *user, unsigned long long *total)
{
    static int f;
    char buff[1024];
    if (f)
    {
        lseek(f,0L,SEEK_SET);
    }
    else
    {
        f = open("/proc/stat",O_RDONLY,0);
        if (f==-1)
        {
            fprintf(stderr,"Gat ekki opna");
            //exit(1);
        }
    }
    int n = read(f,buff,1023);
    if (n<0)
    {
        fprintf(stderr,"Gat ekki lesi (%d)\n",n);
        //exit(2);
    }
    buff[n] = '\0';
    static int num;
    unsigned long long utime,nice,sys,idle,iow,xxx,yyy;
    //printf("Got following line: %s",buff);
    num = sscanf(buff,"%*s %llu %llu %llu %llu %llu %llu %llu %*s",&utime,&nice,&sys,&idle,&iow,&xxx,&yyy);
    *user = utime+nice+sys;
        *total = utime+nice+sys+idle+iow+xxx+yyy;
    //printf("returning %u,%u\n",*user,*total);
}
void printStatus()
{
    debug(1,"main","getting load information");

    // cpu totals
    unsigned long long cpuwork,cputotal;
    getCPUUsage(&cpuwork,&cputotal);

    // program cpu totals
    unsigned long long uuse,suse;
    //getProcessUsage(&uuse,&suse);
    // will be derived after all cpu loads have been fetched

    //reader totals
    unsigned long long rdrCpuUser, rdrCpuSys;
    readPIDStat(pids[PID_READER],&rdrCpuUser,&rdrCpuSys);
```

```
// packet totals
unsigned long packetsRead,packetsTotal;
reader.getCounter(&packetsRead,&packetsTotal);

// filter2
unsigned long long segfiltercpuUser,segfiltercpuSys;
unsigned long segfilterCount, segfilterSize;
readPIDStat(pids[PID_FILT2],&segfiltercpuUser,&segfiltercpuSys);
segfilter.getCounters(&segfilterCount,&segfilterSize);

// filter3
unsigned long long searchcpuUser,searchcpuSys;
unsigned long searchCount, searchSize;
readPIDStat(pids[PID_FILT3],&searchcpuUser,&searchcpuSys);
search.getCounters(&searchCount,&searchSize);

// filterZ
unsigned long long filtZcpuUser,filtZcpuSys;
unsigned long filtZCount, filtZSize;
readPIDStat(pids[PID_FILTZ],&filtZcpuUser,&filtZcpuSys);
filtz.getCounters(&filtZCount,&filtZSize);

// filter4
unsigned long long iplistcpuUser,iplistcpuSys;
unsigned long iplistCount, iplistSize, iplistItems;
readPIDStat(pids[PID_FILT4],&iplistcpuUser,&iplistcpuSys);
iplist.getCounters(&iplistCount,&iplistSize);
iplistItems = iplist.getNumItems();

// other
unsigned long segCount, segMax;
segfilter.getSegCounter(&segCount,&segMax);
double load[3];
getloadavg(&load[0],3);

//   calculated vars
uuse = rdrCpuUser + segfiltercpuUser + searchcpuUser + filtZcpuUser + iplistcpuUser;
suse = rdrCpuSys + segfiltercpuSys + searchcpuSys + filtZcpuSys + iplistcpuSys;

static unsigned long lastTraffic = 0;
double traffic = 0.0;
traffic = (packetsTotal - lastTraffic) / 1024.0;
lastTraffic = packetsTotal;

fprintf(stdout,"LOGGER: %lu %llu %llu %llu %llu %llu %llu %lu %lu %llu %llu %lu %lu %llu %llu,
        %lu %lu %llu %llu %lu %lu %llu %llu %lu %lu %lu %lu %lu %f %0.2f\n",
  time(NULL),                 //   1. current time
  // cpu totals
  cpuwork,                  //   2. working cputime
  cputotal,                 //   3. total cputime
  // program cpu totals
  uuse,                   //   4. total program user cputime
  suse,                   //   5. total program system cputime
  // reader
  rdrCpuUser,               //   r1.reader user cpu
  rdrCpuSys,                //   r2.reader sys cpu
  // packet totals
  packetsRead,                //   6. packet count
  packetsTotal,               //   7. total packet size
  // filter2
  segfiltercpuUser,                 //   8. segfilter user cpu
  segfiltercpuSys,                  //   9. segfilter sys cpu
  segfilterCount,                   // 10. packets in segfilter
  segfilterSize,                  // 11. packet size in segfilter
  // filter3
  searchcpuUser,                  // 12. search user cpu
  searchcpuSys,               // 13. search sys cpu
  searchCount,                // 14. packets in search
  searchSize,               // 15. packet size in search
  // filterZ
  filtZcpuUser,               // 12z. filtz user cpu
  filtZcpuSys,                // 13z. filtz sys cpu
  filtZCount,                 // 14z. packets in filtz
  filtZSize,                // 15z. packet size in filtz
  // filter4
  iplistcpuUser,                  // 16. iplist user cpu
  iplistcpuSys,               // 17. iplist sys cpu
  iplistCount,                // 18. packets in iplist
  iplistSize,               // 19. packet size in iplist
  iplistItems,                // 20. total number of items in iplist
  // segmentation data
  segCount,                 // 21. current seg count
```

```
    segMax,                    // 22. max seg count
    // other
    load[0],                   // 23. 1 min load avg.
    traffic                    // 24. network traffic last 1 sec in KB
  );
  // testing memory
  //struct mstats ms = mstats();
  //printf("memory: bytes_total=%i, chunks_used=%i, bytes_used=%i, chunks_free=%i, bytes_free=%i\n",
  //       ms.bytes_total, ms.chunks_used, ms.bytes_used, ms.chunks_free, ms.bytes_free);
}
void printTable()
{
  debug(2,"main","getting ip table");
  iplist.printTable();
};
void check()
{
  debug(2,"main","->Execute() pid=%i",getpid());
  int count = 0;
  while(1)
  {
    printStatus();
    if (count==15)
    {
      count = 0;
      printTable();
    }
    sleep(1);
    count++;
  }
  printStatus();
}
void printPids()
{
  fprintf(stdout,"pid of main               = %d\n",pids[PID_MAIN]);
  fprintf(stdout,"pid of reader = %d\n",pids[PID_READER]);
  fprintf(stdout,"pid of segfilter                = %d\n",pids[PID_FILT2]);
  fprintf(stdout,"pid of search               = %d\n",pids[PID_FILT3]);
  fprintf(stdout,"pid of filtZ              = %d\n",pids[PID_FILTZ]);
  fprintf(stdout,"pid of iplist               = %d\n",pids[PID_FILT4]);
  fflush(stdout);
}
int main(int argc, char *argv[])
{
  mainpid = getpid();
  pids[0] = mainpid;
  signal(SIGINT, terminate);
  signal(SIGKILL, terminate);
  signal(SIGQUIT, terminate);
  signal(SIGTERM, terminate);
  signal(SIGHUP, reconfigure);
  signal(SIGSEGV, segfault);

  std::set_new_handler(&newFailed);

  fdebug = fopen(debugfile,"w");
  if (fdebug==NULL)
  {
    fprintf(stderr,"Could not open %s",debugfile);
    exit(1);
  }
  // setting default values
  reader.setPacketsCount(5000);
  char *filt0 = "";          // skoa hr taf proxy
  reader.addFilter(filt0);
  reader.setDev("eth0");
  configFile = "filt.conf";

  readCmdLine(argc,argv);
  reconfigure(0);


  debug(1,"main","Starting filters");
  iplist.start();
  search.start();
  filtz.start();
  segfilter.start();
  //filt1.start();
  debug(1,"main","Filters has started");

  debug(1,"main","Starting reader (0x%x)",&reader);
  sleep(1);
```

```
   reader.start(true);
   debug(1,"main","Reader has started");

   sleep(1);
   pids[PID_READER] = reader.getPid();
   pids[PID_FILTZ] = filtz.getPid();
   pids[PID_FILT2] = segfilter.getPid();
   pids[PID_FILT3] = search.getPid();
   pids[PID_FILT4] = iplist.getPid();

   printPids();
   fprintf(stdout,"LOGGER: time cpuwork cputotal uuse suse rdrCpuUser rdrCpuSys pktcnt pktsz f2usr f2sys f2cnt f2sz
           f3usr f3sys f3cnt f3sz fZusr fZsys fZcnt fZsz f4usr f4sys f4cnt f4sz f4itms segcnt segmax load kb/s\n");
   check();

   terminate(0);
   fclose(fdebug);
}


/*
   Project:        Final year project in CS in UNAK
   Author:         biggistefna.is

   File:         bqueue.cpp
   Summary:      The thread-safe queue
*/
#include "bqueue.h"
#include "funcs.h"

BQueue::BQueue(char * n) : name(n)
{
   if (pthread_mutex_init(&mutex,NULL) != 0)
   {
      fprintf(stderr,"Could not init mutex\n");
      exit(1);
   }
   putIndex = getIndex = 0;
   pthread_cond_init(&notFullCond,NULL);
   pthread_cond_init(&notEmptyCond,NULL);
   closed = false;
   char ff[100];
   char *f = ff;
   strcpy(f,n);
   f = strcat(f,".debug");
   fdebug = fopen(f,"w");
   if (fdebug==NULL)
   {
      fprintf(stderr,"Could not open %s",f);
      exit(1);
   }

};

BQueue::~BQueue()
{
   debug(1,name,"Destructing");
   fclose(fdebug);
};

void BQueue::put(void *packet)
{
   debug(2,name,"bq::put now locking");

   if (closed)
      return;

   pthread_mutex_lock(&mutex);

   while (isFull())
   {
      debug(2,name,"bq::put -> waiting");
      fprintf(stderr,"Queue '%s' is full, waiting. time=%lu\n",name,time(NULL));
      pthread_cond_wait(&notFullCond, &mutex);
   }
   array[putIndex] = packet;
   debug(2,name,"bq::putting data (addr:%X) (index=%i)",packet,putIndex);
   putIndex = (++putIndex) % BQUEUE_MAX_SIZE;

   pthread_cond_signal(&notEmptyCond);

   pthread_mutex_unlock(&mutex);
```

```
   debug(2,name,"bq::put unlocked and signaled");
};

void *BQueue::get()
{
   debug(2,name,"bq::get -> trying to lock");
   pthread_mutex_lock(&mutex);

   while (isEmpty())
   {
      debug(2,name,"bq::get -> waiting");
      pthread_cond_wait(&notEmptyCond, &mutex);
   }
   if (closed) return NULL;
   debug(2,name,"bq::getting data (index=%i)",getIndex);
   void *x = array[getIndex];
   array[getIndex] = NULL;
   getIndex = (++getIndex) % BQUEUE_MAX_SIZE;

   pthread_cond_signal(&notFullCond);
   pthread_mutex_unlock(&mutex);
   debug(2,name,"bq::get -> done and unlocked (addr:%X)",x);
   return x;
};

bool BQueue::isEmpty()
{
   return (putIndex==getIndex);
};
bool BQueue::isFull()
{
   return ( ( (putIndex + 1) % BQUEUE_MAX_SIZE ) == getIndex );
};
void BQueue::debug(int d, char *name, char *str, ...)
{
#if DEBUG_B >= d
   if (DEBUG_B>=d)
   {
      va_list argp;
      va_start(argp,str);
      fprintf(fdebug,"DEBUG(%s(%d)): ",name,d);
      vfprintf(fdebug,str,argp);
      fprintf(fdebug,"\n");
      fflush(fdebug);
      va_end(argp);
   }
#endif
}

/*
   Project:       Final year project in CS in UNAK
   Author:        biggistefna.is

   File:       bqueue.h
   Summary:    The thread-safe queue header file
*/

#ifndef _CLASS_BQUEUE
#define _CLASS_BQUEUE
#include <unistd.h>
#include <pthread.h>
#include "netdef.h"

#define BQUEUE_MAX_SIZE 16384


class BQueue
{
   public:
      BQueue(char *);
      ~BQueue();
      void *get();
      void put(void *);
      bool isEmpty();
      bool isFull();
   protected:
      int xxx;
   private:
      void* array[BQUEUE_MAX_SIZE];
      pthread_mutex_t mutex;
      pthread_cond_t notFullCond;
      pthread_cond_t notEmptyCond;
      int putIndex;
```

```
    int getIndex;
    char * name;
    bool closed;
    FILE *fdebug;
    void debug(int , char *, char *, ...);
};
#endif
/*
   Project:       Final year project in CS in UNAK
   Author:        biggistefna.is

   File:        filter.cpp
   Summary:     The super class for the filters
*/

#include <iostream.h>
#include "filter.h"
#include "funcs.h"

Filter::Filter(char *n, BQueue *q) : inQ(q)
{
   init(n);
};

Filter::Filter(char *n, BQueue *iq, BQueue *oq) : inQ(iq), outQ(oq)
{
   init(n);
};
Filter::~Filter()
{
   debug(2,name,"Filter destruction");
   fclose(fdebug);
};
void Filter::init(char *n)
{
   char ff[100];
   char *f = ff;
   f = strncpy(f,n,20);
   f = strcat(f,".debug");
   fdebug = fopen(f,"w");
   if (fdebug==NULL)
   {
      fprintf(stderr,"Could not open %s",f);
      terminate();
   }
   debug(1,n,"Constructing...");
   setName(n);
   threadRunning = true;
   packetsIn = packetSize = 0;
   sem_init(&s_pc,0,1);
};

void Filter::execute()
{
   mypid = getpid();
   debug(1,name,"->execute() myPID = %i",mypid);
   void *packet;
   while(threadRunning)
   {
      packet = inQ->get();
      checkPacket(packet);
   }
};

void Filter::stop()
{
   debug(1,name,"->stop()");
   detach();
   threadRunning = false;
};

void Filter::incCounters(u_long *s)
{
   debug(2,name,"increasing counter");
   sem_wait(&s_pc);
   ++packetsIn;
   packetSize+=*s;
   sem_post(&s_pc);
};
void Filter::getCounters(u_long *c, u_long *s)
{
   debug(2,name,"getting counter");
   sem_wait(&s_pc);
```

```
   *c = packetsIn;
   *s = packetSize;
   sem_post(&s_pc);
};
void Filter::debug(int d, char *name, char *str, ...)
{
#if DEBUG_B >= d
   if (DEBUG_B>=d)
   {
      va_list argp;
      va_start(argp,str);
      fprintf(fdebug,"DEBUG(%s(%d)): ",name,d);
      vfprintf(fdebug,str,argp);
      fprintf(fdebug,"\n");
      fflush(fdebug);
      va_end(argp);
   }
#endif
}
/*
            Project:            Final year project in CS in UNAK
            Author:             biggistefna.is

            File:               filter.h
            Summary:            The generic filter header file
*/

#ifndef _CLASS_FILTER
#define _CLASS_FILTER
#include <unistd.h>
#include "POSIXThread.h"
#include "netdef.h"
#include "bqueue.h"
#include "packet.h"
#include "funcs.h"
#include <semaphore.h>

class Filter : public POSIXThread
{
   public:
      Filter(char*, BQueue *);
      Filter(char*, BQueue *, BQueue *);
      ~Filter();
      void stop();
      virtual void checkPacket(void *) = 0;
      BQueue *inQ;
      BQueue *outQ;
      void setName(char *n) {name = n;} ;
      void getCounters(u_long*, u_long*);
      void getUsage(double *u,double *s) {getRUsage(u,s);}
      int getPid() { return mypid; }
   protected:
      void execute();
      char * name;
      void incCounters(u_long*);
      sem_t s_pc;
      u_long packetsIn;
      u_long packetSize;
      int mypid;
      FILE *fdebug;
      void debug(int , char *, char *, ...);
   private:
      bool threadRunning;
      void init(char *);
};

#endif
/*
   Project:       Final year project in CS in UNAK
   Author:        biggistefna.is

   File:       filtz.cpp
   Summary:    The unzipping filter
*/

#include <iostream.h>
#include "filtz.h"
#include <unistd.h>
#include "funcs.h"
#include "packet.h"
#include "httpitem.h"
#include <zlib.h>
```

```
FiltZ::FiltZ(BQueue *q)
   : Filter("FilterZ",q)
{
};
FiltZ::FiltZ(BQueue *iq,BQueue *oq)
   : Filter("FilterZ",iq,oq)
{
};

FiltZ::~FiltZ()
{
   debug(1,"FilterZ","Destructing. . .");
}


// note: will not check for http encoding
//          must be checked elsewhere (the one that sends it here)
void FiltZ::checkPacket(void *p)
{
   HTTPPacket *hp = (HTTPPacket*)p;
   debug(1,"FilterZ","starting examination packet #%u",hp->packet->number);
   incCounters((u_long*)&hp->packet->size);
   if (hp->packet->size_payload>MAX_UNZIP_SIZE)
   {
      // sennilega best a gera bara hreinlega ekki neitt ?
      // meira a segja sleppa v a skoa lengra
      debug(2,"FilterZ","The packet is too big to unzip => deleting httppacket");
      delete hp;
      hp=NULL;
      return;
   }
   else
   {
      if (unzip(hp)<0)
      {
         debug(2,"FilterZ","could not unzip => so delete httppacket");
         delete hp;
         hp = NULL;
         return;
      }
      outQ->put(hp);
   }
};

/*
   Taken from http://www.zlib.net/zpipe.c
   http://www.zlib.net/manual.html
*/
int FiltZ::unzip(HTTPPacket *packet)
{
   debug(1,"FilterZ","now unzipping packet. Compressed size: %i",packet->packet->size);
   int ret;
   u_char *source, *dest;
   z_stream strm;
   u_long i,compressed_len;
   int wbits = MAX_WBITS;

   // getting full header size (with http)
   u_long headerLengths =          packet->packet->size_ethernet +
         packet->packet->size_ip +
         packet->packet->size_tcp +
         packet->header->headerLength;


   // first copy all headers
   for(i = 0; i < headerLengths;i++)
   {
      bigbuffer[i] = packet->packet->packet[i];
   }


   debug(2,"FilterZ","wrote %i bytes as headerinformation",i);
   // set the dest to the end of the new packet which resides in bigbuffer
   dest = bigbuffer+i;

   source = packet->packet->payload +(packet->header->headerLength);
   compressed_len = packet->packet->size_payload - packet->header->headerLength;

   strm.zalloc = Z_NULL;
   strm.zfree = Z_NULL;
   strm.opaque = Z_NULL;
```

```
strm.avail_in = compressed_len;
strm.next_in = source;
ret = inflateInit2(&strm,wbits);
int initsDone = 1;
if (ret != Z_OK)
{
   debug(2,"FilterZ","Could not initialize the zstream, ret=%i. msg=%s",ret,strm.msg);
   (void)inflateEnd(&strm);
   return ret;
}

while(1)
{
   debug(2,"FilterZ","Now next four chars of source(0x%x) are: 0x%x, 0x%x, 0x%x and 0x%x",
      source,source[0],source[1],source[2],source[3]);
   strm.avail_out = FILTZ_BIG_BUF;
   strm.next_out = dest;
   ret = inflate(&strm,Z_FINISH);

   // taken from tvbuff.c of ethereal code
   if (ret == Z_DATA_ERROR && (*source          == 0x1f) && (*(source + 1) == 0x8b))
   {
      debug(2,"FilterZ","Damn, got Z_DATA_ERROR, so we must do some fix. msg=%s",strm.msg);
      /*
        * inflate() is supposed to handle both gzip and deflate
        * streams automatically, but in reality it doesn't
        * seem to handle either (at least not within the
        * context of an HTTP response.)          We have to try
        * several tweaks, depending on the type of data and
        * version of the library installed.
        */

      /*
        * Gzip file format.          Skip past the header, since the
        * fix to make it work (setting windowBits to 31)
        * doesn't work with all versions of the library.
        */
      Bytef *c = source + 2;
      Bytef flags = 0;

      if (*c == Z_DEFLATED) {
         c++;
      } else {
         debug(2,"FilterZ","sorry, but cannot uncompress gzip");
         (void)inflateEnd(&strm);
         return ret;
      }

      flags = *c;

      /* Skip past the MTIME, XFL, and OS fields. */
      c += 7;

      if (flags & (1 << 2)) {
         /* An Extra field is present. */
         int xsize = (int)(*c |
               (*(c + 1) << 8));

         c += xsize;
      }

      if (flags & (1 << 3)) {
         /* A null terminated filename */

         while (*c != '\0') {
            c++;
         }

         c++;
      }

      if (flags & (1 << 4)) {
         /* A null terminated comment */

         while (*c != '\0') {
            c++;
         }

         c++;
      }
```

```
      inflateReset(&strm);
      compressed_len -= (c - source);
      source = c;
      strm.next_in = source;

      ret = inflateInit2(&strm, wbits);
      ++initsDone;
      debug(2,"FilterZ","OK, lets try that again");
    }
    else if(ret==Z_DATA_ERROR && initsDone<=3)
    {
      debug(2,"FilterZ","Now, lets try if WBITS should be -1, inits:%i",initsDone);
      /*
       * Re-init the stream with a negative
       * MAX_WBITS. This is necessary due to
       * some servers (Apache) not sending
       * the deflate header with the
       * content-encoded response.
       */
      wbits = -MAX_WBITS;

      inflateReset(&strm);

      strm.next_in = source;
      strm.avail_in = compressed_len;

      strm.avail_out = FILTZ_BIG_BUF;
      strm.next_out = dest;
      ret = inflateInit2(&strm, wbits);

      initsDone++;

      if (ret != Z_OK)
      {
        debug(2,"FilterZ","I give up, wasn't WBITS thingy");
        (void)inflateEnd(&strm);
        return ret;
      }
    }
    else if(ret<0)
    {
      debug(2,"FilterZ","Something went wrong with the inflation, ret=%i. msg=%s",ret,strm.msg);
      (void)inflateEnd(&strm);
      return ret;
    }
    else
    {
      debug(2,"FilterZ","Good, we got a good return");
      break;
    }
  }
  (void)inflateEnd(&strm);
  debug(2,"FilterZ","Inflation has ended, inflated %i bytes",strm.total_out);
  debug(2,"FilterZ","-- fyrst 4 chars in uncompressed packet: %c,%c,%c and %c",
    bigbuffer[headerLengths],
    bigbuffer[headerLengths+1],
    bigbuffer[headerLengths+2],
    bigbuffer[headerLengths+3]);
  u_long totalpacketsize = strm.total_out+headerLengths;
  debug(2,"FilterZ","-- last 4 chars in uncompressed packet: %c,%c,%c and %c",
    bigbuffer[totalpacketsize-4],
    bigbuffer[totalpacketsize-3],
    bigbuffer[totalpacketsize-2],
    bigbuffer[totalpacketsize-1]);
  Packet *newpacket = new Packet(bigbuffer,totalpacketsize,packet->packet->number);

  delete packet->packet;
  packet->packet = newpacket;
  return 0;
};

/*
  Project:      Final year project in CS in UNAK
  Author:       biggistefna.is

  File:       filtz.h
  Summary:    The unzipping filter header file
*/

#ifndef _CLASS_FILTZ
#define _CLASS_FILTZ
#include <unistd.h>
#include "netdef.h"
```

```
#include "filter.h"
#include "httppacket.h"
#include "sys/ioctl.h"

#define FILTZ_BIG_BUF 1048576 // 1MB
#define MAX_UNZIP_SIZE 307200 // 300kb

class FiltZ : public Filter
{
   public:
      FiltZ(BQueue *);
      FiltZ(BQueue *,BQueue *);
      void checkPacket(void *);
      ~FiltZ();
   private:
      int unzip(HTTPPacket *);
      u_char bigbuffer[FILTZ_BIG_BUF];
};

#endif
/*
   Project:       Final year project in CS in UNAK
   Author:        biggi stefna.is

   File:          funcs.cpp
   Summary:       general function
*/

#include "funcs.h"
#include <stdio.h>
#include <sys/ioctl.h>
#include "netdef.h"
#include <iostream>
#include <stdarg.h>
#include <fcntl.h>
#include <sys/resource.h>

/*void debug(int d, char *name, char *str, ...)
{
   if (DEBUG_B>=d)
   {
      va_list argp;
      va_start(argp,str);
      fprintf(stderr,"DEBUG(%s(%d)): ",name,d);
      vfprintf(stderr,str,argp);
      fprintf(stderr,"\n");
      va_end(argp);
   }
}
*/
void getRUsage(double *user, double *sys)
{
   struct rusage ruse;
   int res = getrusage(RUSAGE_CHILDREN,&ruse);
   if (res!=0)
   {
      fprintf(stderr,"did not get good ruseage\n");
      ruse.ru_utime.tv_sec = 0;
      ruse.ru_utime.tv_usec = 0;
      ruse.ru_stime.tv_sec = 0;
      ruse.ru_stime.tv_usec = 0;
   }

   *user = ruse.ru_utime.tv_sec + (ruse.ru_utime.tv_usec/1000000.0);
   *sys = ruse.ru_stime.tv_sec + (ruse.ru_stime.tv_usec/1000000.0);
   fprintf(stderr,"getRusage(pid=%d): = %3.6f , %3.6f\n",getpid(),*user,*sys);
};


/*
   Project:       Final year project in CS in UNAK
   Author:        biggi stefna.is

   File:          funcs.h
   Summary:       generic functions
         and the all mighty DEBUG_B constant

*/

#ifndef _DEF_FUNCS
#define _DEF_FUNCS

#define DEBUG_B 0
```

```
#include <time.h>
#include <sys/resource.h>
#include <stdarg.h>
#include <string.h>

//void debug(int, char*, char *,...);
void getRUsage(double *, double *);
#endif
/*
   Project:        Final year project in CS in UNAK
   Author:         biggistefna.is

   File:        header.cpp
   Summary:     The HTTP header class
*/

#include "header.h"
#include <string.h>
#include "funcs.h"
#include <ctype.h>

Header::Header(Packet *p)
{
   hdebug(1,"header","Constructing an header");
   error = 0;
   u_long n = p->number;
   packet = p;
   numItems = 0;
   if (isHeader())
   {
      getItems();
      printItems();
   }
   else
      error = 1;
   hdebug(2,"header","header(%X or packet #%u) constructed error=%i",this,n,error);
};
Header::~Header()
{
   packet = NULL;
   hdebug(1,"header","Deststructing(%X)",this);
};
void Header::printItems()
{
   for(int i=0;i<numItems;i++)
      hdebug(2,"header","#%lu Item[%i]=\"%s\"",packet->number,i,items[i]);
};
bool Header::isHeader()
{
   hdebug(1,"header","isHeader?");
   return (memcmp(packet->payload,"HTTP",4)==0);
};
void Header::getItems()
{
   hdebug(1,"header","starting to get Items");
   u_char *data = packet->payload;
   int i = 0;
   for(u_int c=0;c<packet->size_payload-3;c++) //-3 because of \r\n\0
   {
      hdebug(3,"header","Read char #%i = %c",c,data[c]);
      if ((data[c]=='\r' && data[c+1]=='\n') || data[c]=='\n')
      {
         // end of line
         items[numItems][i] = '\0';
         numItems++;
         if (numItems>MAX_HEADER_ITEMS)
         {
            fprintf(stderr,"header","numItems overflow");
         }
         if (data[c]=='\r') c++;
         hdebug(3,"header","EOL - nextItem=%i",numItems);
         i = 0;
         if ((data[c+1]=='\r' && data[c+2]=='\n') || data[c+1]=='\n' || numItems>=MAX_HEADER_ITEMS)
         {
            // got the whole header
            headerLength = c+3; // run through \r\n (plus 1)
            if (data[c+1]=='\n') --headerLength; // ohh... too far (if header ends in \n\n
            hdebug(3,"header","EOH - end of header. len=%i",headerLength);
            return;
         }
      }
      else
```

```
      {
         items[numItems][i] = data[c];
         i++;
      }
   }
   numItems = 0;
   headerLength = -1;
   error = 1;
   fprintf(stderr,"span through the whole payload... and did not discover the header. some thing is wrong\n");
};

// ath. a er hgt a nota atoi <stdlib.h> til a f"s int r essu
// lka reyndar atol
char *Header::getValue(char *key)
{
   hdebug(1,"header","#%lu now finding the value for %s",packet->number,key);
   if (key==NULL) return NULL;

   int keylen = strlen(key);
   if (keylen>=MAX_IN_LINE) return NULL;

   for(int i=1;i<numItems;i++)
   {
      hdebug(2,"header","#%lu checking %s=%s",packet->number,key,items[i]);
      if (strncasecmp(items[i],key,keylen)==0)
      {
         char *ind = index(items[i],':');
         if (ind!=NULL)
         {
            *ind++;
            while (isspace(*ind))
               *ind++;
            hdebug(2,"header","#%lu found that %s==%s",packet->number,key,ind);
            return ind;
         }
         hdebug(2,"header","#%lu nope that was not the right key (%s!=%s)",packet->number,key,items[i]);
         return NULL;
      }
   }
   hdebug(1,"header","#%lu could not find %s",packet->number,key);
   return NULL;
};
int Header::getResponseValue()
{
   hdebug(1,"header","#%lu now getting response value",packet->number);
   char *item = items[0];
   hdebug(2,"headerResponse","item 0 = %s",item);
   if (item==NULL) return 0;
   //HTTP/x.x 123
   //012345678
   char number[] = "000";
   strncpy(number,&item[9],3);
   int answer = atoi(number);
   hdebug(2,"headerResponse","  answer = %i",answer);
   return answer;
};

void hdebug(int d, char *name, char *str, ...)
{
#if DEBUG_B >= d
   if (DEBUG_B>=d)
   {
      va_list argp;
      va_start(argp,str);
      fprintf(stderr,"DEBUG(%s(%d)): ",name,d);
      vfprintf(stderr,str,argp);
      fprintf(stderr,"\n");
      fflush(stderr);
      va_end(argp);
   }
#endif
}
/*
   Project:       Final year project in CS in UNAK
   Author:        biggistefna.is

   File:       header.h
   Summary:    The http header header file

*/

#ifndef _CLASS_HEADER
#define _CLASS_HEADER
```

```
#define MAX_HEADER_ITEMS 50
#define MAX_IN_LINE 50
#include "packet.h"

void hdebug(int d, char *name, char *str, ...);
typedef char headline[MAX_IN_LINE];
class Header
{
   public:
      Header(Packet *);
      ~Header();
      char * getValue(char *);
      int getResponseValue();
      int error;
      int headerLength;
   private:
      char *header;
      headline items[MAX_HEADER_ITEMS];
      int numItems;
      Packet *packet;
      bool isHeader();
      void getItems();
      void printItems();
};

#endif
/*
   Project:       Final year project in CS in UNAK
   Author:        biggistefna.is

   File:       httpitem.cpp
   Summary:    The http item class
*/

#include "httpitem.h"
#include "funcs.h"

HTTPItem::HTTPItem(HTTPPacket *hp, int p)
{
   //debug(1,"HTTPItem","Constructing..");
   points = p;
   ip_src = (u_long)(hp->packet->ip->ip_src.s_addr);
   number = hp->packet->number;
   size = hp->packet->size;
};
HTTPItem::~HTTPItem()
{
   //debug(1,"HTTPItem","Destructing");
};
/*
   Project:       Final year project in CS in UNAK
   Author:        biggistefna.is

   File:       httpitem.h
   Summary:    The http item header file

*/

#ifndef _CLASS_HTTPITEM
#define _CLASS_HTTPITEM
#include "httppacket.h"

class HTTPItem
{
   public:
      HTTPItem(HTTPPacket *,int);
      ~HTTPItem();
      u_long number;
      int points;
      u_long ip_src;
      u_long size;
};

#endif
/*
   Project:       Final year project in CS in UNAK
   Author:        biggistefna.is

   File:       httppacket.cpp
   Summary:    The http packet
*/

#include "httppacket.h"
```

```
HTTPPacket::HTTPPacket(Packet *p, Header *h)
{
   packet = p;
   header = h;
};
HTTPPacket::~HTTPPacket()
{
   delete header;
   header = NULL;
   delete packet;
   packet = NULL;
};

/*
   Project:       Final year project in CS in UNAK
   Author:        biggistefna.is

   File:       httpacket.h
   Summary:    The http packet header file

*/

#ifndef _CLASS_HTTPPACKET
#define _CLASS_HTTPPACKET
#include "packet.h"
#include "header.h"

class HTTPPacket
{
   public:
      HTTPPacket(Packet *,Header *);
      ~HTTPPacket();
      Packet *packet;
      Header *header;
};

#endif
/*
   Project:       Final year project in CS in UNAK
   Author:        biggistefna.is

   File:       iplist.cpp
   Summary:    The last filter, the IP list manager
*/

#include <iostream.h>
#include "iplist.h"
#include "funcs.h"


IPList::IPList(BQueue *q)
   : Filter("IPList",q)
{
   numberOfItems = 0;
   sem_init(&s_list,SHARED,1);
   sem_init(&s_items,0,1);
   for (u_int i=0;i<MAX_LIST_ITEMS;i++)
   {
      list[i] = NULL;
   }

   debug(1,name,"Construction done");
};

void IPList::stop()
{
   debug(1,name,"stopping");
   detach();
};

IPList::~IPList()
{
   debug(1,name,"Destructing");
   printTable();
   u_long x = getNumItems();
   sem_wait(&s_list);
   for (u_int i=0;i<x;i++)
   {
      debug(2,name,"deleting #%i",i);
      delete list[i];
   }
```

```
   sem_post(&s_list);
};
void IPList::printTable()
{
   debug(1,name,"Printing table:");

   char addr[] = "xxx.xxx.xxx.xxx";
   struct in_addr a;
   u_long x = getNumItems();
   sem_wait(&s_list);
   for (u_int i=0;i<x;i++)
   {
      if (list[i]!=NULL)
      {
         a.s_addr = list[i]->ip_src;
         sprintf(addr,"%s",inet_ntoa(a));
         fprintf(stdout,"List[%i] = {%s,%i,%i}\n",i,addr,list[i]->number,list[i]->points);
      }
   }

   sem_post(&s_list);
};


u_long IPList::getNumItems()
{
   debug(2,name,"getting numItems");
   sem_wait(&s_items);
   u_long x = numberOfItems;
   sem_post(&s_items);
   return x;
};
void IPList::incNumItems()
{
   sem_wait(&s_items);
   ++numberOfItems;
   debug(2,name,"increasing numItems to %u",numberOfItems);
   sem_post(&s_items);
};


void IPList::checkPacket(void *p)
{
   HTTPItem *hi = (HTTPItem*)p;
   debug(1,name,"starting examination packet #%u",hi->number);
   incCounters(&hi->size);
   int i = findInList(hi->ip_src);
   if (i<0)
      add2List(hi);
   else
      updateList(i,hi);
   /*debug(2,name,"Now I'm done using the httpitem, => deleteing it");
   delete hi;
   hi=NULL;*/
};
void IPList::updateList(int index,HTTPItem *hi)
{
   char ip[] = "xxx.xxx.xxx.xxx";
   struct in_addr a;
   a.s_addr = hi->ip_src;
   sprintf(ip,"%s",inet_ntoa(a));

   debug(1,name,"updating the list (%s)",ip);
   sem_wait(&s_list);
   if (list[index]->points < ENOUGH_POINTS)
   {
      list[index]->points += hi->points;
      list[index]->number++;
      debug(2,name,"update => %s => %i points (in %i packets)",ip,list[index]->points,list[index]->number);
   }
   sem_post(&s_list);
};
void IPList::add2List(HTTPItem *hi)
{
   char ip[] = "xxx.xxx.xxx.xxx";
   struct in_addr a;
   a.s_addr = hi->ip_src;
   sprintf(ip,"%s",inet_ntoa(a));

   debug(2,name,"adding to the list (packet #%i (%s): %i points)",hi->number,ip,hi->points);
   u_long x = getNumItems();
```

```
   if (x>=MAX_LIST_ITEMS)
   {
      fprintf(stderr,"the IP List is full");
      return;
   }

   sem_wait(&s_list);
   list[x] = hi;
   list[x]->number = 1;
   incNumItems();
   sem_post(&s_list);
};
int IPList::findInList(u_long ip)
{
   debug(1,name,"Trying to search the list");
   u_long x = getNumItems();
   sem_wait(&s_list);
   int answer = -1;
   for (u_int i=0;i<x;i++)
   {
      if (ip==list[i]->ip_src)
      {
         answer = i;
         break;
      }
   }
   sem_post(&s_list);
   return answer;
};


/*
         Project:          Final year project in CS in UNAK
         Author:           biggistefna.is

         File:             iplist.h
         Summary:          The iplist filter header file
*/

#ifndef _CLASS_FILT4
#define _CLASS_FILT4
#include <unistd.h>
#include "netdef.h"
#include "filter.h"
#include "httpitem.h"
#include <semaphore.h>
#define MAX_LIST_ITEMS 100
#define ENOUGH_POINTS 500
#define SHARED 0

typedef HTTPItem *httpitem;

class IPList : public Filter
{
   public:
      IPList(BQueue *);
      void checkPacket(void *);
      u_long getNumItems();
      void incNumItems();
      void stop();
      void printTable();
      ~IPList();
   private:
      sem_t s_list;
      sem_t s_items;
      httpitem list[MAX_LIST_ITEMS];
      u_int numberOfItems;
      void updateList(int ,HTTPItem *);
      void add2List(HTTPItem *);
      int findInList(u_long );
};

#endif
/*
   Project:       Final year project in CS in UNAK
   Author:        biggistefna.is

   File:       netdef.h
   Summary:    The   file that defines the network packet
*/

#ifndef DEF_NETDEFS
```

```
#define DEF_NETDEFS
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <net/if.h>
#include <netinet/if_ether.h>
#include <netinet/ether.h>
#include <netinet/tcp.h>
#include <netdb.h>

/* Ethernet header */
struct sniff_ethernet {
      u_char ether_dhost[ETHER_ADDR_LEN]; /* Destination host address */
      u_char ether_shost[ETHER_ADDR_LEN]; /* Source host address */
      u_short ether_type; /* IP? ARP? RARP? etc */
};
#define ETH_SIZE 14

/* IP header */
struct sniff_ip {
      #if BYTE_ORDER == LITTLE_ENDIAN
      u_int ip_hl:4, /* header length */
      ip_v:4; /* version */
      #if BYTE_ORDER == BIG_ENDIAN
      u_int ip_v:4, /* version */
      ip_hl:4; /* header length */
      #endif
      #endif /* not _IP_VHL */
      u_char ip_tos; /* type of service */
      u_short ip_len; /* total length */
      u_short ip_id; /* identification */
      u_short ip_off; /* fragment offset field */
      #define IP_RF 0x8000 /* reserved fragment flag */
      #define IP_DF 0x4000 /* dont fragment flag */
      #define IP_MF 0x2000 /* more fragments flag */
      #define IP_OFFMASK 0x1fff /* mask for fragmenting bits */
      u_char ip_ttl; /* time to live */
      u_char ip_p; /* protocol */
      u_short ip_sum; /* checksum */
      struct in_addr ip_src,ip_dst; /* source and dest address */
};
#define IP_SIZE 20
/* TCP header */
struct sniff_tcp {
      u_short th_sport; /* source port */
      u_short th_dport; /* destination port */
      u_long th_seq; /* sequence number */
      u_long th_ack; /* acknowledgement number */
      #if BYTE_ORDER == LITTLE_ENDIAN
      u_int th_x2:4, /* (unused) */
      th_off:4; /* data offset */
      #endif
      #if BYTE_ORDER == BIG_ENDIAN
      u_int th_off:4, /* data offset */
      th_x2:4; /* (unused) */
      #endif
      u_char th_flags;
      #define TH_FIN 0x01
      #define TH_SYN 0x02
      #define TH_RST 0x04
      #define TH_PUSH 0x08
      #define TH_ACK 0x10
      #define TH_URG 0x20
      #define TH_ECE 0x40
      #define TH_CWR 0x80
      #define TH_FLAGS (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
      u_short th_win; /* window */
      u_short th_sum; /* checksum */
      u_short th_urp; /* urgent pointer */
};
#define TCP_SIZE 20

#endif
/*
   Project:       Final year project in CS in UNAK
   Author:        biggistefna.is

   File:       networkreader.cpp
   Summary:    The thread that is responsible for reading the network
```

```
*/

#include <pcap.h>
#include "networkreader.h"
#include <stdio.h>
#include <stdlib.h>
#include "netdef.h"
#include "POSIXThread.h"
#include "funcs.h"

using namespace std;

NetworkReader::NetworkReader(BQueue *q)
{
   fdebug = fopen("reader.debug","w");
   if (fdebug==NULL)
   {
      fprintf(stderr,"Could not open reader.main");
      terminate();
   }
   debug(1,"reader","Constructing a reader...");
   myQ = q;
   packetCount=packetTotal=0;
   count = 0;
   sem_init(&s_pc,0,1);
   debug(2,"reader","Construction is done");
}

NetworkReader::~NetworkReader()
{
   debug(1,"reader","Destructing reader");
   delete packet;
   fclose(fdebug);
}
void NetworkReader::stop()
{
   debug(1,"reader","Stopping reader");
   detach();
};
void NetworkReader::setDev(char *d)
{
   debug(1,"reader","setting Device to %s",d);
   dev = d;
};

void NetworkReader::init()
{
   int op;


   debug(1,"reader","Trying to init... ");

   /*if ((dev = pcap_lookupdev(errbuf)) == NULL)
      error("lookupdev",errbuf);*/
   //dev = "eth0";
   //debug(2,"reader","\tInit->lookupdev(dev=%s) ... done ",dev);


   /* Attach pcap to the network interface */
   // snaplen var BUFSIZE
   if ((handle = pcap_open_live(dev, 1600, 0, 0, errbuf)) == NULL)
      error("open_live", errbuf);
   debug(2,"reader","\tInit->open_live ... done ");

   op = pcap_datalink(handle);
   debug(2,"reader","\tInit->datalink = %i. while DLT_EN10MB = %i",op,DLT_EN10MB);

   if (op != DLT_EN10MB
#ifdef      DLT_IEEE802
        && op != DLT_IEEE802
#endif
        ) error("Check for ethernet, not Ethernet:", dev);

   if ((pcap_lookupnet(dev, &net, &mask, errbuf)) < 0)
      error("lookupnet",errbuf);
   debug(2,"reader","Init->lookupnet... done ");

   if ((pcap_compile(handle, &filter, filter_exp, 0, net)) < 0)
      error("compile",pcap_geterr(handle));
   debug(2,"reader","Init->compile ... done ");

   if ((pcap_setfilter(handle,&filter)) < 0)
      error("setfilter",pcap_geterr(handle));
```

```
   debug(2,"reader","Init: dev=%s",dev);
};
void NetworkReader::setPacketsCount(int n)
{
   if (n<packetCount)
   {
      count = n+packetCount;
   }
   else
   {
      count = n - packetCount;
   }
   debug(2,"reader","setPacketsCount(%i)",count);
};

void NetworkReader::execute()
{
   myPID = getpid();
   debug(1,"reader","READER: starting to exec. pid=%i",myPID);
   init();
   pcap_loop(handle,count,NetworkReader::callback,(u_char*)this);
   debug(2,"reader","closing pcap");
   pcap_close(handle);

}
void NetworkReader::gotPacket(const struct pcap_pkthdr *header, const u_char *p)
{
   debug(2,"reader","gotPacket...");
   if (p==NULL)
   {
      fprintf(stderr,"READER: got null\n");
      return;
   }
   u_long x = incCounter(header->caplen);
   struct sniff_tcp *t = (struct sniff_tcp*)(p + ETH_SIZE + IP_SIZE);
   if (ntohs(t->th_sport) == 80 && header->caplen > MIN_CAPLEN)
   {
      packet = new Packet(p,header->caplen,x);
      debug(2,"reader","Got a big enough(%u) packet from port 80 #%u",header->caplen,x);
      myQ->put(packet);
   }
   else
   {
      debug(2,"reader","packet not sport==80 or not big enough (#%u",x);
   }
}

void NetworkReader::addFilter(char *str)
{
   debug(2,"reader","adding filter %s",str);
   if (strcpy(filter_exp,str)==NULL)
   {
      fprintf(stderr,"Could not copy (%s) to the filter",str);
      terminate();
   }
   //actually does not work :)
}

void NetworkReader::error(char *where, char *err)
{
   printf("ERROR in %s: %s\n",where,err);
   terminate();
}

long NetworkReader::getCounter()
{
   sem_wait(&s_pc);
   long x = packetCount;
   sem_post(&s_pc);
   return x;
}

void NetworkReader::getCounter(u_long *n, u_long *s)
{
   debug(1,"reader","getting Counters");
   sem_wait(&s_pc);
   u_long x = packetCount;
   u_long t = packetTotal;
   sem_post(&s_pc);
   *n = x;
   *s = t;
};
u_long NetworkReader::incCounter(u_long size)
```

```
{
   u_long x;
   debug(1,"reader","increasing counter, sem_wait()");
   sem_wait(&s_pc);
   debug(2,"reader","incCounter, sem_wait() is over");
   x = ++packetCount;
   packetTotal += size;
   sem_post(&s_pc);
   debug(2,"reader","incCounter, sem_post done");
   return x;
};
void NetworkReader::debug(int d, char *name, char *str, ...)
{
#if DEBUG_B >= d
   if (DEBUG_B>=d)
   {
      va_list argp;
      va_start(argp,str);
      fprintf(fdebug,"DEBUG(%s(%d)): ",name,d);
      vfprintf(fdebug,str,argp);
      fprintf(fdebug,"\n");
      fflush(fdebug);
      va_end(argp);
   }
#endif
}

void NetworkReader::callback(u_char *args, const struct pcap_pkthdr *header, const u_char *p)
{
    NetworkReader* r = (NetworkReader*)args;
    r->gotPacket(header,p);
};

/*
          Project:          Final year project in CS in UNAK
          Author:           biggistefna.is

          File:             netdef.h
          Summary:          The network reader header file
*/

#ifndef CLASS_BCAPI
#define CLASS_BCAPI

#include <pcap.h>
#include "POSIXThread.h"
#include "netdef.h"
#include "bqueue.h"
#include "packet.h"
#include "semaphore.h"
#include "funcs.h"

#define MAX_FILT_SIZE 150
#define MIN_PAYLOAD 20
#define MIN_CAPLEN (MIN_PAYLOAD + ETH_SIZE + IP_SIZE + TCP_SIZE)

class NetworkReader : public POSIXThread
{
   public:
      NetworkReader(BQueue *);
      ~NetworkReader();
      void setPacketsCount(int);
      void setDev(char *);
      void addFilter(char *);
      long getCounter();
      void getCounter(u_long *, u_long *);
      void getUsage(double *u,double *s) {getRUsage(u,s);}
      void gotPacket(const struct pcap_pkthdr *, const u_char *);
      void stop();
      int getPid() { return myPID;};
   private:
      void init();
      void actOnPacket();
      u_long incCounter(u_long);
      void error(char *, char*);

      int count;              /* Max number of packets to read */
      long packetCount;       /* To count the packets */
      u_long packetTotal;     /* Total size */

      pcap_t *handle;         /* Session handle */
      char *dev;              /* The device to sniff on */
      char errbuf[PCAP_ERRBUF_SIZE];      /* Error string */
```

```
      struct bpf_program filter;            /* The compiled filter */
      char filter_exp[MAX_FILT_SIZE];          /* The filter expression */
      bpf_u_int32 mask;          /* Our netmask */
      bpf_u_int32 net;           /* Our IP */
      struct pcap_pkthdr header;          /* Packet header */
      BQueue *myQ;
      sem_t s_pc;
      Packet *packet;
      int myPID;
      FILE *fdebug;
      void debug(int , char *, char *, ...);
   protected:
      void execute();
      static void callback(u_char *, const struct pcap_pkthdr *, const u_char *);
};

#endif
/*
   Project:       Final year project in CS in UNAK
   Author:        biggistefna.is

   File:       packet.cpp
   Summary:    The packet
*/

#include "packet.h"
#include "funcs.h"
#include <string.h>
#include <new>

void memFail()
{
   fprintf(stderr,"packet: could not create memory\n");
};

Packet::Packet(const u_char *x, bpf_u_int32 s, u_long n) : number(n)
{
   size = s+1;
#if DEBUG_B > 0
   fprintf(stderr,"packet: (#%lu) constructing a packet(%x) of size: %u\n",n,this,size);
#endif
   size_ethernet = ETH_SIZE;
   size_ip = IP_SIZE;
   size_tcp = TCP_SIZE;

   packet = new u_char[size];
   if (packet==NULL)
   {
      fprintf(stderr,"packet: (#%lu) malloc failed",n);
      return;
   }
#if DEBUG_B > 1
   fprintf(stderr,"packet: (#%lu) have created new memory block (%x)\n",n,packet);
#endif
   for (bpf_u_int32 i=0;i<size-1;i++)
   {
      packet[i] = x[i];
   }
   packet[size-1] = '\0';
#if DEBUG_B > 1
   fprintf(stderr,"packet: (#%lu) has created memory\n",n);
#endif
   ethernet = (struct sniff_ethernet*)(packet);
   ip = (struct sniff_ip*)(packet + size_ethernet);
   tcp = (struct sniff_tcp*)(packet + size_ethernet + size_ip);
   unsigned short offset = tcp->th_off * 4;
   size_payload = size - size_ethernet - size_ip - offset;
   if (size_payload > 0)
      payload = (u_char *)(packet + size_ethernet + size_ip + offset);

};
Packet::Packet(Packet *a, Packet *b)
{
   number = a->number;
   bpf_u_int32 asize = a->size - 1; //-1 for the \0 at the end
   size = asize + b->size_payload ; // we get a terminating null here
#if DEBUG_B > 0
   fprintf(stderr,"packet2: (#%lu) constructing packet(%x) (from two others ([a]=%u,[b]=%u) of size: %u\n",number,this,asize,b->size,size);
#endif
   //debug(2,"packet","Constructing a packet (from two others (a=%x,b=%x) ",a,b);

   //packet = (u_char*)malloc(size);
   packet = new u_char[size];
```

```
   if (packet==NULL)
   {
      fprintf(stderr,"packet: (#%lu) malloc failed",number);
      return;
   }
#if DEBUG_B > 1
   fprintf(stderr,"packet2: (#%lu) have created new memory block (%x)\n",number,packet);
#endif
   bpf_u_int32 i;
   for (i=0;i<asize;i++)
   {
      /*if (a->packet[i]>32 && a->packet[i]<127)
         //printf("%c",a->packet[i]);
      else
         //printf(".");
      */
      packet[i] = a->packet[i];
   }

   for (i=0;i<b->size_payload;i++)
   {
      //printf("%c",b->payload[i]);
      packet[i+asize] = b->payload[i];
   }
#if DEBUG_B > 1
   fprintf(stderr,"packet2: (#%lu) has created memory\n",number);
#endif

   size_ethernet = sizeof(struct sniff_ethernet);
   size_ip = sizeof(struct sniff_ip);
   size_tcp = sizeof(struct sniff_tcp);

   ethernet = (struct sniff_ethernet*)(packet);
   ip = (struct sniff_ip*)(packet + size_ethernet);
   tcp = (struct sniff_tcp*)(packet + size_ethernet + size_ip);
   unsigned short offset = tcp->th_off * 4;
   size_payload = size - size_ethernet - size_ip - offset;
   payload = (u_char *)(packet + size_ethernet + size_ip + offset);
};

Packet::~Packet()
{
#if DEBUG_B > 0
   fprintf(stderr,"packet: (#%lu) Deleting packet (%x) arr=(%x)\n",number,this,packet);
#endif
   delete [] packet;
   packet=NULL;
#if DEBUG_B > 0
   fprintf(stderr,"packet: (#%lu) Deletion successful\n",number);
#endif
};

void packetDump(Packet *packet)
{
   printf("================ Packet number %lu ================\n",packet->number);
   printf("=================== Ethernet ====================\n");
   printf("||\tDestination MAC: %s\n",ether_ntoa((struct ether_addr*)packet->ethernet->ether_dhost));
   printf("||\tSource MAC:                %s\n",ether_ntoa((struct ether_addr*)packet->ethernet->ether_shost));
   printf("||\tType:                      0x%x\n",packet->ethernet->ether_type);

   if (packet->ethernet->ether_type==0x8)
   {
      struct protoent *proto;
      char p[10];
      if ((proto = getprotobynumber(packet->ip->ip_p)) != NULL)
         sprintf(p,"%s (%d)", proto->p_name,packet->ip->ip_p);
      else
         sprintf(p,"%d", packet->ip->ip_p);
      printf("||       ====================== IP ======================\n");
      printf("||       ||\tVersion:               %u\n",packet->ip->ip_v);
      printf("||       ||\tHeader Length:         %u\n",packet->ip->ip_hl);
      printf("||       ||\tType of Service:0x%x\n",packet->ip->ip_tos);
      printf("||       ||\tLength:                %u\n",packet->ip->ip_len);
      printf("||       ||\tID:                    0x%x\n",packet->ip->ip_id);
      printf("||       ||\tOffset:                0x%x\n",packet->ip->ip_off);
      printf("||       ||\tTTL:                   %u\n",packet->ip->ip_ttl);
      printf("||       ||\tProtocol:              %s\n",p);
      printf("||       ||\tChecksum:              0x%x\n",packet->ip->ip_sum);
      printf("||       ||\tSource IP:             %s\n",inet_ntoa(packet->ip->ip_src));
      printf("||       ||\tDestination IP: %s\n",inet_ntoa(packet->ip->ip_dst));

      if (packet->ip->ip_p==IPPROTO_TCP)
```

```
    {
        char flags[64] = "<None>";
        char *fstr[] = {"FIN", "SYN", "RST", "PSH", "ACK", "URG", "ECN", "CWR" };
        int fpos = 0, i;
        unsigned int bpos;
        for (i = 0; i < 8; i++)
        {
            bpos = 1 << i;
            if (packet->tcp->th_flags & bpos)
            {
                if (fpos)
                {
                    strcpy(&flags[fpos], ", ");
                    fpos += 2;
                }
                strcpy(&flags[fpos], fstr[i]);
                fpos += 3;
            }
        }
        flags[fpos] = '\0';

        printf("||       ||   ====================== TCP ====================\n");
        printf("||       ||   ||\tSource Port:           %u\n",ntohs(packet->tcp->th_sport));
        printf("||       ||   ||\tDest. Port:            %u\n",ntohs(packet->tcp->th_dport));
        printf("||       ||   ||\tSeq#:                  %u\n",ntohl(packet->tcp->th_seq));
        printf("||       ||   ||\tAck#:                  %u\n",ntohl(packet->tcp->th_ack));
        printf("||       ||   ||\tData Offset:           %u\n",packet->tcp->th_off*4);
        printf("||       ||   ||\tFlags:                 0x%x (%s)\n",packet->tcp->th_flags,flags);
        printf("||       ||   ||\tWindow size:           %u\n",ntohs(packet->tcp->th_win));
        printf("||       ||   ||\tChecksum:              0x%04x\n",packet->tcp->th_sum);
        if (packet->size_payload>0)
        {
            printf("||       ||   ||   ====================== DATA ====================\n");
            char *prefix = "||       ||   ||   ||\t";
            printf("%s",prefix);
            u_char c;
            const u_char *data = packet->payload;
            int colcount = 0;
            const int LINEMAX = 64;

            for(u_int i=0; i<packet->size_payload; i++)
            {
                c = data[i];
                if (c>32 && c<127)
                    printf("%c",c);
                else
                    printf(".");
                if (colcount==LINEMAX)
                {
                    printf("\n%s",prefix);
                    colcount=0;
                }
                colcount++;
            }
            printf("\n");
            printf("||       ||   ================================================\n");
        }
        printf("||       ================================================\n");
    }

    printf("================================================\n");


};

/*
            Project:            Final year project in CS in UNAK
            Author:             biggistefna.is

            File:               packet.h
            Summary:            The network packet header file
*/

#ifndef _CLASS_PACKET
#define _CLASS_PACKET
#include "netdef.h"

class Packet
{
    public:
        Packet(const u_char *, bpf_u_int32, u_long );
```

```
        Packet(Packet *, Packet *);
        ~Packet();

        u_long number;
        u_char *packet;
        bpf_u_int32 size;

        const struct sniff_ethernet *ethernet; /* The ethernet header */
        const struct sniff_ip *ip; /* The IP header */
        const struct sniff_tcp *tcp; /* The TCP header */
        u_char *payload; /* Packet payload */

        /* For readability, we'll make variables for the sizes of each of the structures */
        u_int size_ethernet;
        u_int size_ip;
        u_int size_tcp;
        u_int size_payload;
};

void packetDump(Packet *packet);

#endif
/*
   Project:        Final year project in CS in UNAK
   Author:         biggistefna.is

   File:         search.cpp
   Summary:      The searching filter
*/

#include <iostream.h>
#include "search.h"
#include <unistd.h>
#include "funcs.h"
#include "packet.h"
#include "httpitem.h"
#include <zlib.h>


Search::Search(BQueue *q)
   : Filter("filter3",q)
{
   init3();
};
Search::Search(BQueue *iq,BQueue *oq)
   : Filter("filter3",iq,oq)
{
   init3();
};
void Search::init3()
{
   debug(1,"Search","Constructing. . .");
   numPatterns=0;
   //getPatterns();
   //compileREs();
   sem_init(&s_config,0,1);
   debug(1,"Search","Construction done");
};

Search::~Search()
{
   debug(1,"Search","Destructing. . .");
   flushPatterns();
   sem_post(&s_config);
}
void Search::flushPatterns()
{
   debug(2,"Search","Flushing patterns (%i)",numPatterns);
   sem_wait(&s_config);
   for(u_int i=0;i<numPatterns;i++)
   {
      debug(2,"Search","freeing reg #(%i)",i);
      regfree(&(regarr[i]));
   }
   numPatterns=0;
   debug(2,"Search","done flushing patterns");
};

void Search::addPattern(char *s)
{
   if (numPatterns==(RE_MAX_COUNT-1))
   {
      fprintf(stderr, "Patterns are full\n");
```

```
      return;
   }
   char b[] = "(^|([[:space:]]|[[:punct:]]|[[:cntrl:]])+)";
   char *p = pattern[numPatterns];

   points[numPatterns] = atoi(s);
   char *f = index(s,' ');
   while (isspace(*f)) *f++;
   int len = strlen(f);
   if (len<1) return;
   if (f[len-1]=='\n')
      f[len-1]='\0';

   if (strcpy(p,b)==NULL)
   {
      fprintf(stderr, "Could not add %s to the pattern begining",b);
      return;
   }
   if (strcat(p,f)==NULL)
   {
      fprintf(stderr,"Could not add %s as pattern number %i",f,numPatterns);
      return;
   }
   debug(2,"Search","Adding pattern: '%s' for %i points",pattern[numPatterns],points[numPatterns]);
   numPatterns++;
};

void Search::compileREs()
{
   debug(1,"Search","compiling regular expressions");
   #define MAX_MESSAGE_LENGTH 1024
   char message[MAX_MESSAGE_LENGTH];
   size_t msize;
   int error;

   for(u_int i=0;i<numPatterns;i++)
   {
      debug(2,"Search","compiling pattern[%i]=%s",i,pattern[i]);
      if ((error = regcomp(&(regarr[i]), pattern[i], REG_ICASE | REG_EXTENDED | REG_NOSUB)))
      {
         msize = regerror(error, &(regarr[i]), message, MAX_MESSAGE_LENGTH );
         fprintf(stderr,"%s\n", message);
         exit();
      }
   }
   sem_post(&s_config);
   debug(2,"Search","compilation of REs done");
};

void Search::checkPacket(void *p)
{
   HTTPPacket *hp = (HTTPPacket*)p;
   sem_wait(&s_config);
   debug(1,"Search","starting examination packet #%u",hp->packet->number);
   incCounters((u_long*)&hp->packet->size);
   if (numPatterns<1)
   {
      fprintf(stderr,"There are no patterns, so, not doing anything");
      sem_post(&s_config);
      return;
   }
   analysePacket(hp);
   sem_post(&s_config);
};
void Search::analysePacket(HTTPPacket *packet)
{
   debug(1,"Search","Analyse Packet #%i",packet->packet->number);
   u_char *p = packet->packet->payload;
   //packetDump(packet->packet);
   int mypoints = getPoints(p);
   HTTPItem *hi;
   if (mypoints>SCORE_THRESHOLD)
   {
      hi = new HTTPItem(packet,mypoints);
      debug(2,"Search","created a new httpItem, forwarding it");
      outQ->put(hi);
   }
   debug(2,"Search","done analyzing the packet, so => delete httppacket");
   HTTPPacket *x = packet;
   packet = NULL;
   delete x;
   x = NULL;
};
```

```
int Search::getPoints(u_char *body)
{
   debug(1,"Search","getting points");
   int error;
   int sum = 0;
   for (u_int i=0;i<numPatterns;i++)
   {
      debug(2,"Search","checking for: %s",pattern[i]);
      error = regexec(&regarr[i], (const char*)body, 1, NULL, REG_NOTEOL);
      if ( error != REG_NOMATCH )
      {
         sum+=points[i];
      }
   }
   debug(2,"Search","Points = %i",sum);
   return sum;
};

/*
            Project:            Final year project in CS in UNAK
            Author:             biggistefna.is

            File:               search.h
            Summary:            The searching filter header file
*/

#ifndef _CLASS_FILT3
#define _CLASS_FILT3
#include <unistd.h>
#include "netdef.h"
#include "filter.h"
#include "httppacket.h"
#include "sys/ioctl.h"
#include <regex.h>
#include <semaphore.h>

#define RE_MAX_COUNT 15
#define PATT_LEN 100
#define SCORE_THRESHOLD 70

class Search : public Filter
{
   public:
      Search(BQueue *);
      Search(BQueue *,BQueue *);
      void checkPacket(void *);
      ~Search();
      void compileREs();
      void flushPatterns();
      void addPattern(char *);
   private:
      void init3();
      void analysePacket(HTTPPacket*);
      regex_t regarr[RE_MAX_COUNT];
      char pattern[RE_MAX_COUNT][PATT_LEN];
      u_int numPatterns;
      int points[RE_MAX_COUNT];
      int getPoints(u_char *);
      sem_t s_config;
};

#endif
/*
   Project:        Final year project in CS in UNAK
   Author:         biggistefna.is

   File:         segfilter.cpp
   Summary:      The segmentation filter
*/

#include <iostream.h>
#include "segfilter.h"
#include <unistd.h>
#include "funcs.h"
#include "header.h"
#include "httppacket.h"

SegFilter::SegFilter(BQueue *q)
   : Filter("segFilter",q)
{
};

SegFilter::SegFilter(BQueue *iq,BQueue *oq,BQueue *zq)
```

```
   : Filter("segFilter",iq,oq)
{
   zipQ = zq;
};
SegFilter::~SegFilter()
{
   debug(1,name,"Destructing");
};
void SegFilter::getSegCounter(u_long *num, u_long *max)
{
   segmenttable.getNumItems(num,max);
};
void SegFilter::checkPacket(void *p)
{
   Packet *packet = (Packet*)p;
   debug(1,name,"starting examination packet #%u",packet->number);
   incCounters((u_long*)&packet->size);
   debug(2,name,"XXX - #%lu pp=%x",packet->number,packet->packet);
   Header *header = new Header(packet);
   debug(2,name,"XXX - #%lu pp=%x",packet->number,packet->packet);
   if (header->error==0)
   { // is an header packet
      int res = header->getResponseValue();
      char *type = header->getValue("Content-Type");
      if (res==200 && type!=NULL && strlen(type)>=9 &&
         strncasecmp(type,"text/html",9)==0)
      { // good, lets dig deeper
         // does the header contain the length parameter?
         char *b = header->getValue("Content-Length");
         debug(2,name,"Content-length==%s",b);
         u_long len = (b==NULL) ? 0 : atol(b);
         debug(2,name,"====> %ld",len);
         if (len==0 || (len<=packet->size_payload && len > 100))
         { // good, we've got the whole packet, or
            // the content-length did not exist
            debug(2,name,"got the whole packet or content-length did not exist = > creating httppacket");
            HTTPPacket *p = new HTTPPacket(packet,header);
            sendPacket(p);
         }
         else
         { // ok, we'll just try to find the rest of packet
            // note: the len for segment is the total payload size
            // therefore, we add the http header (+\r\n\r\n) to len
            debug(2,name,"ohh, got the first segment");
            gotFirstSegment(packet,header,len+header->headerLength+1);
         }
      }
      else
      { // not interested
         debug(2,name,"header was != 200 and != text/html => deleting");
         Packet *x = packet;
         packet = NULL;
         delete x;
         x = NULL;

         if (header!=NULL)
         {
            delete header;
            header = NULL;
         }
      }
   }
   else
   { // does not contain an header
      debug(2,name,"header was currupted, trying to find the header");
      gotSegment(packet);
      if (header!=NULL)
      {
         debug(2,name,"deleting header addr=%X",header);
         delete header;
         header = NULL;
         debug(2,name,"deleting header success");
      }
   }


};
void SegFilter::gotFirstSegment(Packet *packet, Header *header, int len)
{
   debug(1,name,"gotFirstSegment");
   HTTPPacket *p = new HTTPPacket(packet,header);
   // put this in the segmenttable
   segmenttable.addFirst(p,len);
```

```
};
void SegFilter::gotSegment(Packet *packet)
{
   debug(1,name,"trying to find a parent segment");
   int res = segmenttable.add(packet);
   if (res<0)
   {
      // could not add it, so it probably did not belong to
      // other packets
      debug(2,name,"could not find header for this segment => deleting packet(%x)",packet);
      Packet *x = packet;
      packet = NULL;
      delete x;
      debug(2,name,"packet deletion success");
      x = NULL;
   }
   else if (res>0)
   {
      // yes, that was the last segment, so lets forward it
      // but first through away the entry in segmenttable
      debug(2,name,"great, we found the last segment, lets forward it");
      sendPacket(segmenttable.remove(res));
   }
   else
   {
      debug(2,name,"found the parent, waiting for more");
   }
};
void SegFilter::sendPacket(HTTPPacket *packet)
{
   debug(2,name,"sending httppacket");
   char *enc = packet->header->getValue("Content-Encoding");
   debug(2,name,"encoding:%s",enc);
   if (enc!=NULL && strlen(enc)>3 &&
      ((strncasecmp(enc,"gzip",4)==0) ||
        (strncasecmp(enc,"defalte",4)==0)
      ))
   {
      debug(2,name,"found compression encoding... put to zipQ");
      zipQ->put(packet);
   }
   else
   {
      debug(2,name,"found no comp. encoding... put to outQ");
      outQ->put(packet);
   }
};


/*
   Project:       Final year project in CS in UNAK
   Author:        biggistefna.is

   File:       segfilter.h
   Summary:    The segmentation filter header file
*/

#ifndef _CLASS_FILT2
#define _CLASS_FILT2
#include <unistd.h>
#include "netdef.h"
#include "filter.h"
#include "packet.h"
#include "segtable.h"
#include "sys/ioctl.h"
#include "httppacket.h"

class SegFilter : public Filter
{
   public:
      SegFilter(BQueue *);
      SegFilter(BQueue *,BQueue *,BQueue *);
      ~SegFilter();
      void checkPacket(void *);
      void getSegCounter(u_long*, u_long*);
   private:
      void gotFirstSegment(Packet *,Header *, int);
      void gotSegment(Packet *);
      Segtable segmenttable;
      void sendPacket(HTTPPacket *);
      BQueue *zipQ;
};
```

```
#endif
/*
   Project:        Final year project in CS in UNAK
   Author:         biggistefna.is

   File:         segtable.cpp
   Summary:      The segmentation table class
*/

#include <stdio.h>
#include "funcs.h"
#include "segtable.h"

Segtable::Segtable()
{
   firstCheck = 0;
   numItems = maxItems = 0;
   sem_init(&s_items,0,1);
   for(int i=0;i<MAX_SEGITEMS;i++) table[i].free = true;
   fdebug = fopen("segtable.debug","w");
   if (fdebug==NULL)
   {
      fprintf(stderr,"Could not open segtable.debug");
      exit(1);
   }
   debug(1,seg_name,"segtable constructed");
};
Segtable::~Segtable()
{
   debug(1,seg_name,"segtable destructed");
   fclose(fdebug);
};
u_long Segtable::getNumItems()
{
   debug(2,seg_name,"getting counter");
   sem_wait(&s_items);
   u_long x = numItems;
   sem_post(&s_items);
   return x;
};
void Segtable::getNumItems(u_long *n, u_long *m)
{
   debug(2,seg_name,"getting counter and max");
   sem_wait(&s_items);
   *n = numItems;
   *m = maxItems;
   sem_post(&s_items);
};
void Segtable::incNumItems()
{
   debug(2,seg_name,"increasing counter");
   sem_wait(&s_items);
   ++numItems;
   if (numItems>maxItems)
      maxItems=numItems;
   sem_post(&s_items);
};
void Segtable::decNumItems()
{
   debug(2,seg_name,"decreasing counter");
   sem_wait(&s_items);
   --numItems;
   sem_post(&s_items);
};
void Segtable::addFirst(HTTPPacket *packet, int len)
{
   debug(1,seg_name,"adding the first packet of length %i",len);
   debug(2,seg_name,"adding First: Packet->number=#%i, size=%i",packet->packet->number,packet->packet->size);
   unsigned int i = firstCheck;
   unsigned int count = 0;
   debug(2,seg_name,"remember true=%i,false=%i",true,false);
   while(1)
   {
      debug(2,seg_name,"checking if table[%i] is free %i",i,table[i].free);
      if (table[i].free)
      {
         debug(2,seg_name,"ohh, great found a free table(%i)",i);
         table[i].packet = packet;
         table[i].len = len;
         table[i].free = false;
         firstCheck=(++i%MAX_SEGITEMS);
         incNumItems();
```

```
            return;
        }
        i++;
        if (i>=MAX_SEGITEMS)
            i = 0;
        count++;
        if (count>=MAX_SEGITEMS)
            break;
    }
    // now, we've overflown the table :(
    fprintf(stderr,"SegmentTable overflown\n");
    // XXX-for now, just don't add it :s
};

// return >=0 for success
//             >0 for last packet (is the index of that item)
//             <0 for not found
int Segtable::add(Packet *packet)
{
    debug(1,seg_name,"adding segment to parent");
    int i = find(packet);
    if (i<0)
        return i;
    debug(2,seg_name,"well, we found a matching packet i=%i. #%i",i,table[i].packet->packet->number);
    Packet *newpacket = new Packet(table[i].packet->packet,packet);
    debug(2,seg_name,"constructed a new packet of size %i, table.len=%i",newpacket->size_payload,table[i].len);
    Packet *x = packet;
    packet = NULL;
    delete x;
    x = NULL;
    delete table[i].packet->packet;
    table[i].packet->packet = newpacket;
    int rest = table[i].len - table[i].packet->packet->size_payload;
    debug(2,seg_name,"rest of packet is %i",rest);
    if (rest==0)
    {
        // return i+1 because of the return thingy for this function
        debug(2,seg_name,"so, returning %i",i+1);
        return i+1;
    }
    else
    {
        if (rest<0)
        {
            debug(2,seg_name,"overshot the length, returning %i",rest);
            return rest;
        }
        else
        {
            debug(2,seg_name,"still some left (rest=%i)",rest);
            if (rest < PACKET_REST_ALLOW)
            {
                debug(2,seg_name,"but it is less than %i, so we stop here",PACKET_REST_ALLOW);
                return i+1; //+1 because of return thingy
            }
            return 0;
        }
    }
};
int Segtable::find(Packet *packet)
{
    debug(1,seg_name,"trying to find a parent");
    int i = firstCheck;
    int count = 0;
    while(1)
    {
        // TODO - maybe, put ip_dst and th_sport also ?
        if (!table[i].free &&
            (table[i].packet->packet->tcp->th_ack == packet->tcp->th_ack) &&
            (table[i].packet->packet->ip->ip_src.s_addr == packet->ip->ip_src.s_addr) &&
            (table[i].packet->packet->tcp->th_dport == packet->tcp->th_dport)
        )
        {
            debug(2,seg_name,"found parent i=%i",i);
            return i;
        }
        i++; // ++i%=MAX_SEGMENTS ??
        if (i>=MAX_SEGITEMS)
            i = 0;
        count++;
        if (count>=MAX_SEGITEMS)
            break;
    }
```

```
   debug(2,seg_name,"nope did not find the parent");
   return -1;
};

HTTPPacket * Segtable::remove(int index)
{
   debug(1,seg_name,"removing an item of index: %i",index-1);
   table[index-1].free = true;
   decNumItems();
   return table[index-1].packet;
};
void Segtable::debug(int d, char *name, char *str, ...)
{
#if DEBUG_B >= d
   if (DEBUG_B>=d)
   {
      va_list argp;
      va_start(argp,str);
      fprintf(fdebug,"DEBUG(%s(%d)): ",name,d);
      vfprintf(fdebug,str,argp);
      fprintf(fdebug,"\n");
      fflush(fdebug);
      va_end(argp);
   }
#endif
}
/*
            Project:              Final year project in CS in UNAK
            Author:               biggistefna.is

            File:                 segitem.h
            Summary:              The segmentation table header file
*/

#ifndef _CLASS_SEGDEF
#define _CLASS_SEGDEF
#define MAX_SEGITEMS 1000
#define PACKET_REST_ALLOW 10
#define seg_name "segtable"
#include "httppacket.h"
#include <semaphore.h>

struct segitem
{
   HTTPPacket *packet;
   int len;
   bool free;
};

class Segtable
{
   public:
      Segtable();
      ~Segtable();
      int add(Packet *);
      void addFirst(HTTPPacket *, int);
      HTTPPacket *remove(int);
      void getNumItems(u_long*, u_long*);
      u_long getNumItems();
   private:
      struct segitem table[MAX_SEGITEMS];
      unsigned int firstCheck;
      int find(Packet *);
      sem_t s_items;
      u_long numItems;
      u_long maxItems;
      void incNumItems();
      void decNumItems();
      FILE *fdebug;
      void debug(int , char *, char *, ...);
};

#endif
```