A Tool For Online Packet Analysis In Mobile Networks

NAVID FARADJZADEH

Master's Degree Project
Stockholm, Sweden June 2012

XR-EE-LCN 2012:007
MASTER THESIS REPORT

A Tool For Online Packet Analysis In Mobile Networks

NAVID FARADJZADEH

Supervisors

GYÖRGY DÁN (KTH)           HÅKAN TRANBERG (ERICSSON AB)

June 13, 2012      Gothenburg, Sweden
Preface

This work is a Master of Science thesis project that was done at Ericsson AB and the result of the project is going to be used at Ericsson packet core SGSN-MME support unit in Lindholmen-Gothenburg.
Acknowledgements

On the way to realize an idea into a working example one faces enormous number of difficulties that takes a great deal of knowledge and experience. As much as there is satisfaction and content on the way there are times of frustration and disappointment as well. All this was too much for me to take without the support and advice from wonderful people around me for whom I want to take the opportunity to express my deepest gratitude and respect.

From Ericsson AB, I want to thank my supervisor Håkan Tranberg whose brilliant ideas helped the project develop so far. Håkan with his experience and indisputable knowledge put me on the right track on my way to realizing this thesis. I also want to thank Claes Uggla whose suggestions helped me integrate into the environment and make the best out my experience at Ericsson AB.

I want to thank Görgy Dán my supervisor at KTH who patiently walked with me throughout the entire work from the first day to the last. His precise, critical and brilliant view on my work is what has made this thesis worth reading.

My family, specially my parents and my brother Vahid were the ones who backed me up always and under any condition. Without their full support, I would not be at this point in my life and doing this thesis in the first place.

Finally, It is needless to mention that I am also thankful to many people supporting me with their advices and ideas in this work from my colleagues at Ericsson to my friends and I regret that I cannot mention them one by one here.
Abstract

Today the mobile networks face a high demand for packet data delivery at ever increasing rates. In order to satisfy the demand, new network modules and protocols are introduced into mobile networks. This leads to a complex system of protocols and algorithms. This level of complexity requires efficient methods of troubleshooting. This need has motivated the implementation of more efficient packet analizers besides the ones that already exist today.

A number of packet analizers have already been introduced in order to read the data from the wires and dissect them into packets but the process of reading binary data and dissecting them is a multistage procedure. In this project we tried to propose a more efficient solution to packet analizing in mobile networks domain. Packet analizers that already exist can dissect a packet provided that it is a full packet or in case that there is an absent header the user must be aware of the type of the header and add it to the packet manually and then hand it over to a packet analizer for dissection. However, here in this project we implemented a packet analizer that can dissect any headers extracted from a packet without the need to have prior information on the type of the absent headers.

In this report we describe the types of protocols that our solutions potentially can support then we discuss the requirements and constraints of such a tool. We give a description of the design and implementation of the software and finally we discuss some improvements on the performance of our solution.
Contents

1 Introduction ......................................................... 10
  1.1 Methodology .................................................. 11
  1.2 Structure .................................................... 12

2 Mobile Internet and Design Issues ................................. 13
  2.1 Mobile Radio Generations 0 and 1 .......................... 14
    2.1.1 Cellular Mobile System ................................. 14
  2.2 Second Generation(2G) ..................................... 15
    2.2.1 GPRS .................................................. 17
  2.3 Third Generation(3G) ...................................... 17
    2.3.1 Universal Mobile Telecommunication System(UMTS) ... 18
    2.3.2 Core Network ......................................... 18
      2.3.2.1 Gateway GPRS Support Node (GGSN) .............. 18
      2.3.2.2 Serving GPRS Support Node (SGSN) .............. 19
  2.4 Forth Generation(4G) and beyond ......................... 19
    2.4.1 Long Term Evolution(LTE) ............................. 20
      2.4.1.1 Mobility Management Entity(MME) .............. 20
    2.4.2 SGSN-MME support ................................... 20
      2.4.2.1 SGSN-MME and Protocols ......................... 21
      2.4.2.2 Support Unit .................................... 21
  2.5 Wireshark .................................................. 22
    2.5.1 Text2pcap ............................................. 23
    2.5.2 Tshark .............................................. 23

3 Online Packet Analizer ........................................ 24
  3.1 Requirements ............................................... 24
    3.1.1 Query Submission Method ............................. 25
    3.1.2 Protocol Support ..................................... 25
    3.1.3 Database ............................................. 25
    3.1.4 Output Interface ..................................... 26
  3.2 Constraints ................................................ 26
    3.2.1 Processing Complexity ................................ 26
    3.2.2 Code Complexity ..................................... 27

4 Design and Implementation ...................................... 28
  4.1 User Submission ........................................... 28
  4.2 Processing The Input ...................................... 29
  4.3 Displaying The Results .................................... 30
4.4 Login System .................................................. 30
4.5 Database ........................................................... 30
4.6 Handling Files .................................................... 30
4.7 GUI Design of OPA .............................................. 32

5 Performance Improvements .............................. 37

6 Conclusion ......................................................... 46
List of Figures

2.1 Cellular Mobile Network ........................................ 15
2.2 GSM protocol architecture .................................... 16
2.3 GPRS protocol stack ........................................... 17
2.4 3G Core Network ................................................. 19
2.5 E-UTRAN overall architecture ............................... 20
2.6 The S1-MME protocol stack ................................... 22
2.7 A wireshark screenshot .......................................... 23

4.1 OPA input form snapshot ........................................ 33
4.2 OPA results page snapshot ...................................... 34
4.3 Login window ..................................................... 35
4.4 Database update form ............................................ 35
4.5 Entry check page .................................................. 35
4.6 Procedures list page ............................................. 36
4.7 HTML view .......................................................... 36

5.1 Cumulative number of system calls for uniformly distributed input 38
5.2 Cumulative number of system calls for biased input ....... 39
5.3 Cumulative number of system calls for same type inputs . 40
5.4 Cumulative number of system calls for same type inputs in the long run ........................................ 40
5.5 Comparison of the performance of Hard and Soft priority in response to slow and fast changing input pattern 42
5.6 Comparison of the performance of Hard and Soft and Dynamic priority in response to slow and fast changing input pattern 43
5.7 Realistic Scenario .................................................. 44
List of Tables

5.1 Comparison of user experience in the biased input type . . . . . . 44
5.2 Comparison of user experience in the same type input . . . . . . 44
5.3 Comparison of user experience in the same type input . . . . . . 45
Acronyms

BS Base Station
CDMA Code Division Multiple Access
CGI Common Gateway Interface
CN Core Network
CS Circuit Switched
EDGE Enhanced Data rates for GSM Evolution
eNB E-UTRAN Node B
EPC Evolved Packet Core
ETSI European Telecommunications Standards Institute
E-UTRA Evolved UMTS Terrestrial Radio Access
E-UTRAN Evolved UMTS Terrestrial Radio Access Network
FDMA Frequency Division Multiple Access
GERAN GSM/EDGE Radio Access Network
GSM Global System for Mobile communications
GGSN Gateway GPRS Support Node
GMM GPRS Mobility Management
GPRS General Packet Radio Service
GTP GPRS Tunnelling Protocol
GUI Graphical User Interface
HLR Home Location Register
HSDPA High-Speed Downlink Packet Access
HSUPA High-Speed Uplink Packet Access
HTML HyperText Markup Language
HTTP Hyper Text Transfer Protocol
IM IP Multimedia
IMSI International Mobile Subscriber Identity
IMT International Mobile Telecommunication
IPTV TV through the internet
IPv4 Internet Protocol version 4
ITU International Telecommunication Union
LTE Long Term Evolution
MME Mobility Management Entity
MMS Multimedia Messaging Service
MS Mobile Station
MSC Mobile Switching Centre
MSC Mobile-services Switching Centre
NAS Non-access Stratum
PDN Packet Data Network
PDP Packet Data Protocol
PS Packet Switched
PSTN Public Switched Telephone Networks
P-TMSI Packet-Temporary Mobile Station Identity
RNC Radio Network Controller
SCTP Stream Control Transmission Protocol
SGSN-MME Serving GPRS Support Node and Mobility Management Entity
SGW Serving GateWay
SM Session Management
SMS Short Messaging System
S1AP S1 Application Interface
TCP Transport Control Protocol
TDMA Time Division Multiple Access
UE User Equipment
UMTS Universal Mobile Telecommunication System
VLR Visitor Location Register
WAP Wireless Application Protocol
WWW World Wide Web
3GPP 3rd Generation Partnership Project
Chapter 1

Introduction

A packet analizer – or a packet sniffer – is a tool that can be installed on some network nodes and can read all the binary data passing through the node’s interfaces and save them onto the memory. It can parse the bits into meaningful packets with various network headers later from the memory or in real time and on the go. This process is rather complex, since there are enormously large number of protocols that are actively in use in today’s networks. To get a feeling of how many protocols might be out there just consider that starting from link Layer in TCP/IP protocol suite and only for Ethernet there is a field (etherType) to indicate the next layer protocol and this field is 16 bits long which means there potentially can be 65536 different next layer protocols and among these potential protocols IPv4 is only one. In the IPv4 header there is another field (Protocol) that indicates the next layer protocol. This field itself is 8 bits long which indicates 256 different potential protocols for the next layer. The applications that TCP can carry on top or application layer are indicated by the port numbers in TCP header. Since there are application protocols that can use any free ports and moreover these applications can be designated to carry higher level protocols there can be literally infinite number of application layer protocols. HTTP is an example of an application layer protocol which works on port 80. This was only TCP/IP protocol suite. Taking for example mobile internet into account opens up another world of protocols.

These countless protocols each having their own data structure and encoding suggests that the task of packet analyzing is quite a complex process. But the good news is these protocols all share a common characteristic. They all have a means inside them to indicate what next layer protocol they are carrying. In other words we can start from link layer and after dissecting link layer header we will already have known what next header protocol we are expecting. This can reduce packet analyzing tools design into implementing specific dissectors for each protocol\(^1\) then upon finding out what is to be expected next just call the proper dissector for the next protocol.

\(^1\)As there are enormous number of protocols this can only be realized in the best form by distributed collaboration of many people. These kinds of collaborations being the heart of open source society, best packet analyzer softwares you can find are products of open source. Wireshark is the perfect example of such rich softwares that is realized this way.
As long as the analyzer tool sits right behind the network interface everything is fine. We have a complete packet hex dump and we have the characteristics of the network card hence enough information about the first layer and then we only need to start dissecting. But there comes serious obstacles in the whole process when it comes to real world applications of such a tool. Different Network industry corporations might not follow the regulations indicated in RFCs. For example etherType in Ethernet header of Ericsson SGSN-MME product might not be 0x0800 but still carrying IPv4 protocol as the next header. Fortunately packet sniffers generally have options for these purposes so that they can be forced to interpret the next header as IPv4 regardless of etherType value for example. In another scenario a hex dump might be handed to packet analyzer that starts with IP for instance rather than Ethernet. This also can be handled by introducing fake headers. The problem can be even more complicated. A hex dump might be handed to a packet sniffer for which there is no former information about what header is absent. This later problem is not solved in today’s packet analyzers since for such class of problems there is no straightforward solution. The hex dump might start with any number and this number might represent the first byte of any header. Here in our project we tried to find an optimized solution to this later problem for a specific application domain (Mobile Networks).

This project is a design and implementation of an online packet analyzer which is capable of dissecting any hexdumps uploaded provided that the type of top level protocol is indicated. Despite the fact that the process of dissecting is straightforward from bottom to top; The reverse process, which is dissecting from top to bottom seems to be an intensive task. Along with this capability this web service contains a database of sample network procedures that will be used for network support and maintenance purposes.

1.1 Methodology

In the beginning, this project was just an idea. During the first week I got familiar with Perl scripting and CGI programming and did a little research on Wireshark and generally packet dissection algorithms in order to see the possibilities that these tools as well as Perl CGI programming can provide. It was then that I had an estimate of the extent and potential possibilities of the software that I could implement in regard to the limited time for implementation. In one month I could form a solid knowledge of Perl CGI scripting and implement a prototype to see how the software would look like. During this I had meetings with my supervisor to define and decide on the requirements of the software. In the second month I became more familiar with the protocol types and hex dumps that support staff deals with mostly. An important part of the project was to find out what protocols should be added. As I did not have an extensive knowledge of all the protocol types in SGSN-MME I had to add these new protocols based on the requests from support staff. This made me implement the code in a generic style so that I could integrate new protocols support in the shortest time possible. At the end of the second month the beta version of the software was ready. During the third month I tried to optimize the header guessing algorithm and draw diagrams and tables to be able to discuss the issues in the algorithm. In the forth month I had a first version ready to
be used and I had a meeting with potential users of the software in order to introduce the software and let them use it in order to find out further bugs and problems with the software.

1.2 Structure

The second chapter is a very short introduction of mobile internet networks and some well known protocols in that area since this project is going to be used in a network support unit for SGSN-MME which is a very central module in today’s mobile internet. We will take a look at the challenges ahead in order to introduce fake headers into mobile internet protocols. In the third chapter we will introduce our online packet analyzer tool and later we will discuss its design and implementation in chapter 4. Fifth chapter is a discussion on the efficiency of our header guessing method and finally in chapter 6 we will discuss the future works and possibilities of this tool.
Chapter 2

Mobile Internet and Design Issues

"Imagine wherever you have mobile phone coverage you are also connected to internet; A connection just as fast as your connection at home."

The idea is fairly simple and the problem — at least on the surface — is clear. In fact Mobile Internet is no more than what the name suggests; We want Internet and we want it to be mobile. Meaning that we want to be connected while we are carrying our Internet enabled device; However, the main problems of today’s designs arise when we want to connect exactly the same way as we are connected through fixed internet connections; With the same quality and speed. There are numerous challenges ahead while trying to design and implement such a system like, Service Quality, Availability, Low price, etc...

In fact mobile radio networks in the beginning were to transmit audio over radio channels and they were not intended to be used for data communication let alone Internet connectivity. The main concern in mobile radio was to utilize a radio channel bandwidth to carry audio signals along with some control signals. As we will see through coming sections the need for data transmission gradually started to emerge. From the introduction of SMS in 2G to fully packet switched Internet connectivity in 4G we will see how mobile infrastructure starts to evolve in order to cope with this need.

Here in this chapter we will try to cover engineering efforts that led to today’s mobile internet technology and possibly the future of it. In fact the history of such efforts can be summarized to Zero, First, Second, Third, Forth and Fifth generations of mobile telephony. The point that should be mentioned is that throughout this chapter by interfaces between to modules we mean protocols designed specifically for these devices in order to communicate. Interfaces can be considered as languages between two specific network devices. A large amount of application layer protocols are designed for these interfaces.
2.1 Mobile Radio Generations 0 and 1

0G is the first effort to have wireless communication. It was started after the second world war with introduction of mobile telephone service through which calls were set up by a wireless operator. Cell connectivity had not been introduced yet.

In fact in the beginning the use of designated protocols was not as extensive and complex as it is today. Technologies used in 0G were PTT (Push to Talk), MTS (Mobile Telephone System), IMTS (Improved Mobile Telephone Service), AMTS (Advanced Mobile Telephone System), OLT (Norwegian for Offentlig Landmobil Telefoni, Public Land Mobile Telephony) and MTD (Swedish abbreviation for Mobiletelefonsystem D, or Mobile telephony system D)[1]. These are not protocols as we know them today but algorithms in order to multiplex and manage the sending and receiving different voice channels with the least loss and distortion over radio carriers. Quite a few number of radio channels were used in such systems.

1G was introduced after a number of standards developed in 1980's[1]. It was the beginning of cellular mobile radio which is the base of today's mobile radio design. 1G networks were still analogue and used to transfer voice using FDMA technique. Here we try to expand on cellular networks since it is the foundation of today's mobile networks.

2.1.1 Cellular Mobile System

In order to have a cellular network four important items are necessary as illustrated in Figure 2.1[3],

1. Mobile station
2. Base station
3. Mobile switching centre
4. Public switched telephone network (PSTN)[2]

Mobile Station is the device that connects the subscriber to the base station. An MS is composed of a user interface between the subscriber and the system and a radio unit which is able to connect to the base station[2]. The user interface is commonly composed of a keypad along with a speaker and a microphone. As the mobile telephony has advanced in time this interface has also become more and more complicated. A common example of an MS is a mobile handset or nowadays known as Cellphone.

Base Station is responsible to take the signal from an MS in its respective cell coverage and transmit it to the switching centre via land line. This type of transfer is done on a circuit switched basis. There is a bandwidth which is divided into channels using techniques such as FDMA/TDMA and the whole portion of the bandwidth is allocated to this transmission. There is no packets involved but rather chunks of data or frames that are sent in time/frequency slots.

Another important task in a BS\(^1\) is to detect the signal strength to recommend handoff if necessary[2]. The handoff is an important technique that was

\(^1\)Base Station
developed to make the moving of mobile station with the least loss of quality possible. There are two methods to transfer these type of control signals. One way is to dedicate specific bandwidth exclusively to transfer control signals another way is to insert silent periods in between the voice frames and transmit these control signals inside those frames. This type of signalling is done much more efficiently in packet switched networks such as the internet using packet protocols. As the need for data transmission increases the need for control signalling also increases and this type of signalling as in 1G seems not to be the most efficient in terms of utilization and bandwidth.

Mobile Switching Centre is responsible to interface the mobile units and the fixed telephone network. It switches the calls coming from a number of BS’s. An MSC service area is the number of BS’s that it can serve. A PSDN is the conventional telephone network that treats the MCS as fixed telephones[2].

2.2 Second Generation(2G)

2G was a cellular telephony system that was installed based on the GSM standard in Finland in 1991. Second generation is not designed for data communication, however SMS was introduced into it. It is used in 212 countries and by almost 2 billion people around the world[1]. It uses two technologies namely CDMA and TDMA. 2G was aimed to provide an increased voice clarity and reduced power needed by mobile systems to operate on it. GSM architecture follows the layerd OSI model. There are three major layers in GSM architecture[4],

1. **Layer 1**: Physical Layer which is radio transmission.

2. **Layer 2**: Data Link Layer for error free transmission.

3. **Layer 3**: Networking Layer to transfer call related messages between various network entities.

In Figure 2.2 the general GSM protocol stack along with the Interfaces between different network components can be seen.
In parallel to these improvements, the Internet on the other hand was traditionally based on packet switched networks. This packet switched characteristic of the Internet facilitated employing enormous number of Protocols therefore the range of applications it could support was incomparable to Mobile Networks. In an attempt to modulate and carry digital data such as packets over conventional PSTN analogue systems and also to demodulate them in the digital receivers modems were introduced. However, this is a point where Mobile Networks start to implement packet switched functionality besides their circuit switched nature in an attempt to support wider range of applications. 2.5G systems came after introduction of packet switched domain in addition to circuit switched domain. GPRS implements packet switched domain by the ability to establish and maintain virtual channels rather than physical channels in order to transmit packet data. This kind of connection accesses the radio channel only when a packet is being transmitted otherwise the physical medium is released to be used by other connections while the virtual connection is not disrupted and only goes idle and as soon as the next packet arrives the connection starts to utilize the channel again. This type of connection is more efficient than conventional circuit switched connections that establish a physical connection and occupy the channel during the whole connection. However voice and SMS transmission in 2.5G is still done using circuit switched connections. 2.5G was a step toward 3G systems[1]. 2.5G systems are officially defined as GPRS (General Packet Radio Service) technology. GPRS supporting packet switched transmission, could be used for applications such as WAP, MMS, WWW and Email. Such packet switched networks needed well defined protocols in the form of data chunks each accompanied with a proper header with addresses and ports defined, therefore the need for packet analyzing in mobile networks started to emerge.

In an attempt to reach higher data rates EDGE was introduced into GSM systems. EDGE is in fact an enhancement on GSM systems which can be implemented on any GPRS enabled Network provided that the carrier implements upgrades needed[1].
2.2.1 GPRS

GPRS is an attempt to add packet delivery functionality to conventional GSM networks. All the network elements in GSM needed to be modified in order to be able to support packet connectivity[13]. There are two major enhancements to GSM in GPRS which is adding two new modules for packet connectivity - SGSN and GGSN. The mobile networks with GPRS suddenly could support a wide range of protocols. The protocol stack in GPRS is demonstrated in Figure 2.3[13].

![GPRS protocol stack](image)

As is depicted in Figure 2.3 a mobile station can connect to the Internet using IP connectivity. This connection is made through GGSN that works like a gateway and connects the internal mobile network to outside packet data networks like the internet. The GTP or GPRS Tunneling Protocol is a protocol introduced in GPRS in order to make and maintain a tunnel from mobile station to the nearest GGSN for internet connectivity. We will talk about this protocol later in coming sections.

2.3 Third Generation(3G)

3G emerged fulfilling the ITU IMT-2000 specifications[1]. 3G is based on a number of standards such as UMTS – standardized by 3GPP – and CDMA2000 – standardized by 3GPP2. It makes use of TDMA and CDMA and is compatible to work with 2G systems. The advancements of 3G over 2.5G includes support for a wider range of applications such as, Video-conferencing, Web and WAP browsing at higher speed and IPTV support.

Such applications needed their respective protocols in order for their packets to be transmitted in mobile networks. The introduction of HDSDPA into 3G network makes the 3.5G. It’s a mobile telephony protocol which is a smooth evolutionary path towards UMTS. The introduction of HSUPA is the start of UMTS which is a quite evolved version of 3G[1].
2.3.1 Universal Mobile Telecommunication System (UMTS)

UMTS was standardized by European Telecommunication Standards Institute (ETSI). Later 3GPP took the role of standardizing 3G. Radio Access Network, Core Network, Terminals, Services and System Aspects and GERAN are the 5 important 3GPP standardization areas. A UMTS network consists of three interacting domains,

- Core Network (CN);
- UMTS Terrestrial Radio Access Network (UTRAN);
- User Equipment (UE);

A UE is the same as MS in 1G except it should be 3G enabled. In UTRAN the base station is called Node-B and control equipment for Node-B is called RNC. The core network is based on GSM with GPRS. Here we take a very brief look at core network and its elements.

2.3.2 Core Network

The Core Network is divided to PS\(^2\) (Packet Switched) domain, CS\(^3\) (Circuit Switched) domain and IM subsystem\(^5\). The most important core network elements are illustrated in Figure 2.4\(^{11}\).

The Home Location Register (HLR) is the location register to which a mobile subscriber is assigned for record purposes such as subscriber information.\(^5\)

The Mobile-services Switching Centre (MSC) constitutes the interface between the radio system and the fixed networks. The MSC performs all the necessary functions in order to handle the circuit switched services to and from the mobile stations.\(^5\)

A Radio Network Controller (RNC) is a network component in the PLMN that implements the functions of controlling of one or more Node Bs. Node B is a logical network component which serves one or more UTRAN cells.\(^5\)

The other two very important modules are SGSN and GGSN for which we will have a short discussion.

2.3.2.1 Gateway GPRS Support Node (GGSN)

GGSN is a PS domain module that provides connectivity to external Packet Data Network (PDN) – such as the Internet – using Gi interface. Interfaces such as Gi in core network are groups of protocols used in order to transmit specific data between two core network modules or to transmit information to outer networks. GGSN can be considered as a typical IP router that implements additional functionality to support mobile connectivity. Such functionality includes GPRS Tunnelling Protocol (GTP). GTP can dynamically adjust the tunnel according to user’s mobility.\(^6\) It is carried on top of SCTP.

\(^2\)A “PS type of connection” transports the user information using autonomous concatenation of bits called packets: each packet can be routed independently from the previous one.\(^5\)

\(^3\)A “CS type of connection” is a connection for which dedicated network resources are allocated at the connection establishment and released at the connection release.\(^5\)
2.3.2.2 Serving GPRS Support Node (SGSN)

SGSN is a key core network element which provides PS related user-plane and control-plane functions. In control plane these functions include GPRS Mobility Management (GMM) and Session Management (SM) functions.

The SGSN stores two types of subscriber data for handling originating and terminating packet data transfer; GMM and SM information. When a mobile attaches to a PS i.e. to one SGSN, the SGSN establishes a GMM for that mobile. Furthermore if a mobile establishes a new PS bearer the SGSN creates a new PDP Context.

The GMM context includes the mobile’s permanent identity (IMSI), it’s temporary identity for the PS domain (Packet-Temporary Mobile Station Identity, P-TMSI), the address of the Visitor Location Register (VLR) currently serving the MS, information for authenticating the MS, etc.

The PDP context includes information on an established PS bearer, such as PDP address, the PDP type, the external PDN related to this PS bearer, etc.[6]

2.4 Forth Generation (4G) and beyond

4G is basically the extension in 3G technology with more bandwidth and services offered. The expectation for 4G technology is basically the high quality audio/video streaming over end to end Internet Protocols[1]. Internet protocols in this technology will be the main method of connectivity delivering an almost fully packet switched connectivity for mobile devices built upon the mobile infrastructure. LTE standard in fact was the step toward 4G mobile networks.

5G Technology stands for 5th Generation Mobile technology. 5G technology has changed the means to use cell phones within very high bandwidth. Nowadays mobile users have much awareness of the cell phone (mobile) technology. The 5G technologies include all types of advanced features which makes them most powerful and in huge demand in near future[1]. The discussion of 5G
technological expectations is beyond the scope of this report.

2.4.1 Long Term Evolution (LTE)

LTE was developed in 3GPP and more formally is referred to as Evolved UMTS Terrestrial Radio Access (E-UTRA) and Evolved UMTS Terrestrial Radio Access Network (E-UTRAN). One of the most important objectives of LTE is to implement an all IP network in which the whole connectivity in the network is packet based [7].

![E-UTRAN overall architecture](image)

As can be seen from Figure 2.5 [9] an E-UTRAN is composed of eNBs. eNBs are interconnected with X2 interfaces. They are connected to EPC or more specifically to MME (Mobility Management Entity) with S1 interface [8].

2.4.1.1 Mobility Management Entity (MME)

MME is the key control-node for the LTE access-network. It is responsible for idle mode UE (User Equipment) tracking and paging procedure including retransmissions. It is involved in the bearer activation/deactivation process and is also responsible for choosing the SGW for a UE at the initial attach and at the time of intra-LTE handover involving Core Network (CN) node relocation. It is responsible for authenticating the user [10].

2.4.2 SGSN-MME support

SGSN-MME (Serving GPRS Support Node and Mobility Management Entity) is a core network module with different responsibilities like maintaining a session for mobile connectivity to Internet during the period between opening and closing a connection. It uses different routing algorithms in order to send and receive packets through its interfaces. SGSN-MME uses some number of interfaces for IP connectivity. Setting up these interfaces uses different protocols and methods. Here in this chapter we try to make a brief description of these interfaces along with the protocols involved in it. Later we will move on to SGSN-MME support and give a short introduction of the nature of their work.
2.4.2.1 SGSN-MME and Protocols

As mentioned before there are numerous interfaces through which the core network elements are connected to each other. In order to setup these connections, a number of interfaces are introduced in SGSN-MME. These interfaces include:

- **Gn Interface** to connect SGSN-MME to GGSNs and other SGSN-MMEs within the same PLMN.
- **Gb Interface** to connect to BSCs.
- **Gp Interface** to connect SGSN-MME to GGSNs and SGSN-MMEs in the other PLMNs.
- **Gr Interface** to connect SGSN-MME to Home Location Registers (HLR).
- **S1-MME Interface** to connect SGSN-MME to eNodeBs which are in turn connected to UE.

These are only some examples and there are many such interfaces. In order to implement these interfaces, a large number of specific protocols are designed. For instance, when a user wants to start a session, SGSN creates a *PDP Context* on behalf of the user. The MS sends a request to join a PDN. SGSN-MME, after authenticating it, will establish a virtual data network between that MS and the PDN. SGSN-MME uses the GTP (GPRS Tunnelling Protocol), an application protocol to set up and dynamically change the settings of the MS connection to the PDN. S1-MME interface as another example is an application layer protocol that connects SGSN-MME to eNodeB. S1AP (S1 Application Protocol) is the type of messages transferred in this connection and NAS (Non-Access Stratum) are transferred between SGSN-MME and UE. S1-MME transfers S1-AP messages over SCTP (Stream Control Transmission Protocol). SCTP itself is another transport layer protocol. S1AP being an application protocol carries on top NAS Messages which is another application protocol. In Figure 2.6 you can see an overview of the different protocols used to interface MME only.

There are many protocols involved in SGSN-MME alone. SGSN-MME is a mobile network module that connects mobile nodes to the internet. Taking into account all the protocols in the Internet today helps to imagine how diverse the input types to a packet analyzer can be.

2.4.2.2 Support Unit

In SGSN-MME support unit, engineers should find the faults and bugs in the module. The debugging process requires them to perform extensively the task of dumping the hex bytes from interfaces and inspect them packet by packet to find the anomalies in regular network procedures. There are devices to dissect these bytes in massive scale — and as they are being generated — but it happens quite a lot of times that these bytes cannot be dissected properly therefore they appear in trace files as hex bytes rather than meaningful packets. The situation gets critical when there is an emergency call and they have to find a fault that happened in a module that is put to work for a customer. In such circumstances

---

4Starting a session can be considered as connecting to the Internet

5Picture taken from Ericsson internal documents
a tool that can take these bytes and do the cumbersome task of guessing absent headers and dissecting the bytes into correct packet representation is of great benefit.

2.5 Wireshark

With the emergence of packet switched technology in mobile networks there is an increasing demand for powerful packet analyzing methods and tools in this area. There are some number of packet analyzers in use today. tcpdump\(^6\), netsniff-ng\(^7\), Wireshark\(^8\) and a lot of other packet sniffers\(^9\).

We preferred Wireshark for this work for some reasons,

- Wireshark's license is BSD GPL\(^10\) which means it is free and open source.
- Wireshark is a software with a lot of capabilities and features and a comfortable GUI that we have imitated in our online web form for the sake of integrity. Furthermore a huge number of programmers from around the world are still working on it and updating it for the newest technologies and protocols.
- Wireshark is also being updated inside Ericsson and by Ericsson developers for the new technologies that Ericsson is using.

A screenshot of Wireshark GUI is depicted in Figure 2.7\([12]\). Wireshark includes in its package some number of programs.

\(^6\)http://www.tcpdump.org/
\(^7\)http://netsniff-ng.org/
\(^8\)http://www.wireshark.org/
\(^9\)In Wikipedia there is an article about this under the title of “Comparison of packet analyzers”
\(^10\)For more info on free licences check http://www.gnu.org/licenses/license-list.html
2.5.1 Text2pcap

In order for Wireshark to work you have to feed it with a proper file format i.e. “.pcap”. text2pcap converts a properly formatted text file with all the hex bytes in it into a capture file that can later be fed to Wireshark.\textsuperscript{11} text2pcap is a part of Wireshark distribution and is available in Wireshark download package.

2.5.2 Tshark

Tshark can be considered as the engine of Wireshark. Tshark can print the results in terminal as well as generating them in all kinds of useful file formats.\textsuperscript{12} In other words Wireshark is enhanced tshark with a rich GUI. As we have implemented our own HTML web form we only need Tshark rather than Wireshark.

Tshark takes as input a “.pcap” file format and generates the results in the form of a number of file formats like,

- Text; Which is a human readable text of the summary of the packet. We used this to out put the summary of the packet in our form.
- psml; Packet Summary Markup Language. An XML format of the summary of the packet information.
- pdml; Packet Details Markup Language. An XML format of the Detailed information of the packet as is displayed in Wireshark output. We used this to generate an HTML page containing a wireshark-like GUI with all the packet details.

\textsuperscript{11}\url{http://www.wireshark.org/docs/man-pages/text2pcap.html}
\textsuperscript{12}\url{http://www.wireshark.org/docs/man-pages/tshark.html}
Chapter 3

Online Packet Analizer

In network support and maintenance as the nature of their work implies, there is an extensive need for analyzing packets in the network. A big part of what they do is to dump the raw hex data from connectors and see actually what is going on. These dumps are of course not easy to read for humans and they need to be dissected into human readable text. There is a number of Packet Analysers that are used today for this purpose. We mentioned some of them in previous chapter. They listen to the network interfaces and sniff the data and display them in packet format. They can also take an input file of previously dumped data in a proper format (e.g. “.pcap” for Wireshark) and display them just the same way. All packet analyzers come as software packages and can be installed and used in PCs. However the core of this project is to implement an online packet analyizer which can be accessed through web. We have also added various features to this online packet analyizer.

3.1 Requirements

OPA is supposed to be a web page i.e. a URL through which you can access the services. The first idea in the beginning was to have a server that could receive a stream of hex data and analyze it and dissect it into human readable packet representation and return the result back. Furthermore the server should be able to return some relevant information about the packet. The requirements were almost clear and straightforward but there was nothing more on how it should look like or how should it operate nor any framework in which the software should be implemented. After discussions and a little research the following minimum requirements were decided,

1. User friendly input interface; The server should be able to receive queries through an easy to use interface.

2. Wide range of protocol support and flexible ability to dissect any kind of meaningful hex stream; The server should be able to dissect the hex stream into correct packet representation and announce the results as detailed as wireshark does.
3. Flexible smart and intense database of side material; The server should be able to keep a database of the correct examples of network procedures and return those correct examples in a useful setting beside the results.

4. User friendly output interface; The server should be able to represent the output in an easy to read, use and understand manner.

It is notable that these are minimum tasks that the server at least should be able to perform. Of course there are many possibilities for adding extra features to the system.

3.1.1 Query Submission Method
The user query submission should be very easy to use. The scenario would be like the user is looking at a hex dump and he/she needs to know what it represents on the go. Therefore it will only take a copy and paste of the data from any screen into the text box in input form and submit it. In fact it is possible to copy and paste any piece of text and the software will search for any relevant patterns of hex bytes and tries to dissect them provided that it is indicated if the data has offset numbers or it is just raw hex dump. This input method was developed having that in mind that if the user encounters a hex stream he/she will want to know what it represents so he/she might just copy the text from any console and paste it. The software should be as robust as possible since in many cases the hex stream is embedded inside a lot of other irrelevant text. Therefore looking or a hex stream and copying it might be cumbersome for human eye.

The update database part should be easy to use and it should take all the relevant information in an easy way. There should be a part to enter a short description of what the database entry is about and it should accommodate the capture sample file to be assigned to the entry. It should also be quite easy to delete an entry.

3.1.2 Protocol Support
One important issue is the hex dumps that are uploaded to the web page. These hex dumps are generated using all kinds of packet sniffing devices and later have been manipulated in all kinds of ways. For instance some hex dumps are selected from an already dissected packet data and there is no useful information of what they represent or what kinds of headers are stripped off. The software should be flexible enough to add all the necessary information to the hex dumps make a packet that can be dissected and display it.

3.1.3 Database
When a support engineer debugs a bogus procedure he/she does not have in mind the correct sequence of all the packets involved in the procedure so it will be of a great use if a correct example of the possible procedures that involve that packet be displayed along with the dissected packet. This raises the need to have a database. These procedures are numerous and highly complex. Furthermore everyday might rise a need for a new procedure to be added to the database.
Therefore the database should be dynamic rather than a static repository of sample capture files.

In order to manipulate and update this database we need a login system to allow trusted users to add or remove or change these samples. These users should be aware of what type of description they assign to the procedures so that they can later be searchable using proper tags. Such type of administered access to database is a direct result of the need to update the database regularly. There is no sign up form for this log in system since we do not want to give the privilege of administration to more than 3 users at a time. Therefore the sign up procedure would be to send a request to the maintainer of the code and he/she would manually add the required username and password to the access list.

3.1.4 Output Interface

The output should be as easy to use as Wireshark. In other words the packet representation should be displayed in a way that users can fold and unfold the details and study all the elements in the packets easily. It should also display some samples of the procedures that might involve this type of protocol. These information should be easy to access and there should be a download part so that users can download their capture file that was generated during the process for later use.

The database should have comfortable search options to browse through the procedures. It should be able to display the procedures in HTML format as well as the ability to download the capture file into the local hard drive.

3.2 Constraints

There are a number of constraints in dealing with the problem. Some constraints rise from the very nature of the problem and some of them are the result of the environment in which the software is supposed to operate. The first thing to consider in any design is that what kind of platforms are we expecting our software to run on. In our project we have a server which runs on Linux\(^1\). The web server is Apache\(^2\). Users should be able to connect to the server from their systems from all kinds of OS or web browsers. In our design we tried to use the most controversial HTML and Perl features so that it can be compatible with almost all systems and browsers. In some cases we used some features that was added later to aforementioned languages.\(^3\)

3.2.1 Processing Complexity

This service will be run on a server along with many other processes. Therefore we have a limited processing power that we can utilize. Therefore we should keep the processing complexity as low as possible. Reading the input text for example could be smarter but it would put huge burden on the processing complexity

\(^1\)More specifically we started with a Red hat Linux server only for development purposes. We will move on to another system the soon our software is ready to be used.

\(^2\)Apache 2.2.3 but we will move to another system

\(^3\)Like use of iFrame in HTML that was added later and was not supported by many browsers long ago but we believe most of today’s browsers support them all.
of the program that we cannot tolerate. Even after extracting the proper hex stream there are bigger issues in processing complexity. As was mentioned in the introduction we need to guess the write headers to be added to the hex stream so that it can be dissected. Designing an algorithm to guess these information is processor intensive as well. Time complexity of such method is rather high and it appears to be the bottleneck of our system. However in chapter 5 we take a deeper look at the method we used for improving time complexity of this method.

We will have to access the hard drive many times during the operation. Tshark and text2pcap read from files on the hard drive and therefore we have to save the results of every step onto the hard drive in order for them to be usable by these softwares. This amount of access to the hard drive is not the most efficient and reliable but again we can tolerate it since the number of these accesses are linear and they do not grow with the size of input.

3.2.2 Code Complexity

The code for this program needs constant maintenance. As the next person to maintain the code could be some one other than the coder himself the code flows should be easy to understand. However the nature of some algorithms used in this project requires complex coding. This complexity might cause the software to be hard to maintain. However, we tried to use clear variable names and easy to understand code flows to make it as simple as possible to trace. Furthermore we even sacrificed some features for the sake of simplicity of the design. For example updating the HTML form dynamically or showing the results at the same page as the input form made the design quite complex while it was not really crucial for the requirements.
Chapter 4

Design and Implementation

We chose Perl\(^1\) CGI scripting to write the program since all the other Programs
and functions on the server are implemented with Perl, so for the sake of integrity
we chose to write in Perl. Furthermore since there are many scripts in the
web server running on Perl we already had a rich collection of libraries and
interpreters installed on the web server for Perl. Perl is also a very powerful
tool for handling text (As we need it in our project) through regular expressions
and string manipulation functions. Our whole design can be summarized into
the following stages,

1. User submission.
2. Processing the input and generating the results.
3. Displaying the results.
4. Login system.
5. Database search and update.

4.1 User Submission

The flow of the code for accepting and processing the user queries and displaying
them is as the following

\[
\text{deleteOldfiles()};
\text{if (defined $uploadedFileName) }
\{
    \text{getTheUploadedFile()};
\}
\text{else}
\{
    \text{getTheBytes()};
\}
\]

\(^1\)http://www.perl.org/
First we check if there is a file uploaded or the bytes are pasted into text box. After retrieving them we store them and go to next step which is to format input and prepare it to be fed into tex2pcap. In this step a file with extension ".raw" is created to keep the original user input. Here in getTheUploadedFile() we save the file in "raw" or in function gettheBytes() save the text. In get function getTheBytes() we run the algorithm to extract the meaningful hex stream from the input text. In this step we need to know if the bytes are raw hex dumps or offsetted bytes. The result of this step is kept on the hard drive inside a file with extension ".fmt". This file is the proper format to feed into tex2pcap. Next step is to convert the file into ".pcap" format.

### 4.2 Processing The Input

In this step we need to generate capture file and — using that capture file — to generate actual output in form of packet representation. Imagine that user has uploaded a hex dump of SCTP header and all the other headers are striped off already. In order to make a proper ".pcap" file we need to add a fake IP header as well as a fake Ethernet header on top of it. So what we need is to call tex2pcap with an option to add fake IP header like the following,

```
text2pcap -i 132 $inputFile $outputFile;
```

"-i 132" in the above command means that text2pcap should add an IP header which encapsulates an SCTP header. That explains the need to have the radio button list and prompt the user to choose the type of protocol.

But as mentioned before we still do not know which header is absent. This is handled in convertToPcap function. In order to deal with this we start with testing the hex dump by adding headers one after another and see which one dissects the packet as we expect. Imagine the SCTP scenario again. Upon receiving the input we have to try it as it is first. If the result from tshark is not relevant to SCTP we will add Ethernet header and try it again if the result was not relevant we add IP and so on. There are some considerations on how to prioritize these headers and which header to add first in order to converge to a solution as fast as possible. We will take a look at these methods in discussion.

During this process a ".pcap" is created and the actual packet is generated with tshark. The only thing we need to do is to save it in a text file. Saving the result is done inside ReportBack() function. We make a ".txt" file of the summary packet in report back as well as generating a ".pdml" file. All these files will be published in the results page for the user to download them.

---

2 text2pcap will add the fake Ethernet header automatically
4.3 Displaying The Results

The function `htmlAnounce()` makes an HTML form and displays all the information generated. We use the “.pdml” file in order to display the results in a wireshark like HTML format. We have a `pdml2html` file containing a Java script to transform our PDML file into an HTML web page to display in output screen. For every query there is a pdml file created in XML format. In order to display it we copy this pdml to the html\textsuperscript{3} folder of the server. Then call it upon display. There is a small consideration when we make the pdml file name by appending the time of the day as well as the process id. This ensures a unique file name in case that there are couple of queries in close succession, eliminating the danger of two processes writing into the same file. We also use the packet summary generated with tshark and display it. All warning messages are also made and displayed in the output form.

4.4 Login System

There are numerous network procedures and they are evolving as the new technologies appear. Therefore we need a method so that privileged users can add and delete sample capture files of these procedures from the database. In order to handle administered update of the database we have made a login system. This is a login page that takes the username and password then matches it with a list of username password pairs to allow access. This list is updated manually and there is no sign up system. This list is not coded or scrambled in any way and it is plain text. In turn we highly control the access to this file. This page is only one CGI file and it redirects to itself in cases that there is a problem with login. Problems such as no username or password provided or incorrect username and password combination.

4.5 Database

The Database update flow is quite straightforward. The update form appears after login. It checks if the entry is already submitted then adds it to the database. This update consists of saving the sample capture file into the hard drive then adding entry’s name along with the description and tags assigned to it inside a file.

In order to generate tags we use the procedure name provided by the user. Then we tokenize the description and extract the words and use them as tags as well. We exclude the very common words\textsuperscript{4} from the tag list so that the procedure will not be included in irrelevant search queries.

4.6 Handling Files

In order to make files we have appended system time with “seconds” precision. One issue in handling files is that if two consecutive requests are sent to the server and two instances of the script are run in less than one second then the

\textsuperscript{3}Or it can be htdocs folder as well

\textsuperscript{4}These words are called Stop Words in the literature
file names are the same resulting in two processes writing into same files. In order to solve this we have appended the process ID along with system time to the files to make sure that files generated by different instances of the script have different names. Taking this approach in the design we can be sure that each instance of the script makes files that are unique to that instance only and therefore is secure from other scripts making changes to them.

There are a lot of files generated during the running of the program. We can delete the files instantly after the execution of the script but as mentioned in requirements part, we need to keep them for a little while so that the users have time to download their results file. These files pile up and occupy a large portion of the hard drive after a while. Therefore we need to get rid of them occasionally. We decided to remove the files that are older than 15 minutes. There are couple of approaches to do so.

- To write a Perl script to delete files and set up the system\(^5\) to run that script in every 15 minutes.
- To write a Perl script and make it run forever and delete the files every 15 minutes.
- To make the main script to handle the job somehow.

The problem with the first approach is that we become dependent on system resources that we cannot trust thoroughly. In real world servers there are many restrictions and updates of the policies and restarts that you cannot rely on the on-schedule running of your script. Furthermore, on the servers that different people are running different jobs you are never sure if your scheduling is secure.

The second approach does not have the problem of being dependent on the system resources but on the other hand you are keeping a process running for ever hence misusing system resources most of the time. This approach is also not secure to interruptions by third parties.

We took the third approach as we believe it is the most robust way of doing such a thing. As someone sends a request to the server the script is run once, we check for all the files and if they are e.g. older than 15 minutes in our case we delete them in the very start of the script. In this manner we make sure when a user starts its job on the server there are no old files there. This approach has the problem of not deleting the files if no one visits our web page. But we need not to worry about that since once the script is invoked again it will make sure to clean up the hard disk.

This system can be modeled as an \(M/G/\infty\) queue. A queue with Poisson arrivals and a general distribution of the service time and infinite number of servers. Assume that the holding time for every file is \(T\). Therefore the service time for every job is \(T\). In other words \(\mu = 1/T\). The arrival process is Poisson with rate \(\lambda\). Our actual number of servers – slots available on the hard drive – here is \(K\) rather than \(\infty\). Therefore we are fine as long as there are \(k \leq K\) files in the queue. We would like the probability of our server being in state \(P_k\) or above be less than 0.01. This way we can guarantee the longest \(T\) for keeping the files on the disk while the probability that the disk gets full is less than 0.01. In other words we want \(\sum_{k=0}^{K} P_k \geq 0.99\) at all times.

\(^5\)In Linux setting up a “cron job” will do that.
To estimate $\lambda$ at $n^{th}$ invocation we can consider the exponentially smoothed measure of the interarrival time $t$ between two consecutive invocations from the following formula,

$$t_n = \alpha t + (1 - \alpha)t_{n-1}$$  

(4.1)

$\alpha$ is the smoothing weight that can be 0.01 in our case. Now at the $n^{th}$ invocation we have $\lambda_n = 1/t_n$. As $K$ is not a very well defined value we have to estimate it. From the different input sizes considered in many test queries an average value of 3.7 MB per query could be a good estimate. Using this estimate and considering $B$ as the total memory size considering $K = B/3.7$ would be a relatively good estimate.

On the other hand, we know the steady state probability that the queue is in state $k$ is,

$$P_k = \frac{\rho^k}{k!}e^{-\rho}$$  

(4.2)

Therefore we have

$$\sum_{0}^{K} \frac{\rho^k}{k!}e^{-\rho} \geq 0.99$$  

(4.3)

We solve this inequality for $\rho = \lambda/\mu$ which in turn can be solved for $\mu = 1/T$. Hence if $\mu \geq m$ therefore $T \leq 1/m$. Considering the left hand side of 4.3 as $B(K)$. We can also calculate $B(K)$ using the following recursion,

$$B(K) = (1 + \frac{\rho}{K})B(K - 1) - \frac{\rho}{K}B(K - 2)$$  

(4.4)

Where,

$$B(0) = e^{-\rho}$$  

(4.5)

$$B(1) = (1 + \rho)e^{-\rho}$$  

(4.6)

### 4.7 GUI Design of OPA

Online Packet Analyzer is a web page with the Interface as depicted in Figure 4.1. There are 2 methods of uploading input to the website. To copy and paste the hex data as text into the text box provided in the form or to upload a text file containing the hex data using upload file section in the form are the two methods of query submission. All the characters other than hex numbers will be ignored from input. The interpreted characters as input data are “0” to “9” and “a/A” to “f/F”. Many delimiters can be inserted in between the actual hex data(e.g. delimiters like , or ;) freely since they will be ignored. No need to mention that if one inserts one of the before mentioned hex representation characters inside the hex stream they will be counted as relevant input hex data and hence leading to erroneous results. The input will be scanned for relevant hex streams offsetted or raw. The radio buttons at the bottom of the page are to choose between raw or offsetted format. These streams will then be forwarded to be processed. We have put the option to upload a file containing hex dumps as it is quite common to have the hex dumps on a text file. Imagine the user has received a hex dump in a text file. It is quite comfortable to upload the whole file at once. The user also have to specify what top level protocol her/his hex bytes represent. There are protocol options to choose from on the list at
The user can also request the results to be sent to his/her Email by providing an Email address at the bottom of the page. After submitting the input the packet representation of the hex dump will be shown as a result. As is illustrated in Figure 4.2 at the top there is a wireshark-like list of the details of the packet. The user can browse through the details by opening and closing the tabs in the form. In the text box under the wireshark window a summary of the packet is provided for a quick look. At the bottom of the page there are links to download some types of output files that were generated from the results. The user can download a plain text file containing packet summary. It contains exactly the same text shown in the text box. The capture file is also available in case one wants to open it in wireshark. A PDML file is also available which is the XML representation of the packet details. The referenced hex stream can also be downloaded which can be used as input to text2pcap program. At the bottom of the packet details window there is a warning message. It indicates that the Ethernet header is a fake header. In fact if any header was added by the program there will be a warning indicating it. We discussed fake headers in previous sections.

From main form the login system can be accessed to login to database. The login system as is shown in Figure 4.3 gives the advantage of manipulating the database to qualified users.

After logging into database there will be a form like in Figure 4.4 through which the database can be updated. First item in the form is a name to choose.

---

6The need to specify the protocol will be discussed further in coming sections.
for the procedure to be uploaded. The text box below that, is a short description of the procedure. The name along with the description are used to generate tags to make this procedure searchable. The last item is a “.pcap” file that has to be uploaded to be assigned to the entry.

After submitting a summary of the entry is displayed as is shown in Figure 4.5 to be confirmed.

After confirming the submission the page will be directed to the procedures list as is shown in Figure 4.6. The database consists of a collection of all the possible procedures in the network. For instance the can be a capture file that includes the complete procedure for LTE network Attach. Here the view can be filtered using search queries. Beside each element in the list there are three options. The option to delete the procedure from the list or to download the capture file or to simply view the capture file in HTML format. These procedures can be returned along with and in relevance with the byte streams results uploaded by the user. Upon clicking view HTML the user will be directed to a page where the procedure’s — as is shown in Figure 4.7 — HTML view can be seen.
Figure 4.3: Login window

Figure 4.4: Database update form

Figure 4.5: Entry check page
Here is a summary of the procedures in the database

Figure 4.6: Procedures list page

Figure 4.7: HTML view
In the previous chapter we mentioned an important constraint of our system: The fact that we do not know which header is absent. Taking a closer look we can see that there can be at least 5 different absent header types,

1. No Header is absent
2. Link Layer Header is absent
3. Network Layer Header is absent
4. Transport Layer Header is absent
5. Application Layer Header is absent

The naive approach to the problem is that regardless of any extra information, every time we check all the headers in the same order.\(^2\). Remember that for each try we have to have at least 2 system calls which is a very time consuming process. Therefore avoiding any system calls can save us a lot of run time.

One way to improve the performance is to consider somehow the history of the headers submitted before, hoping that any upcoming queries might probably be of the type that has been uploaded most. In order to investigate the effectiveness of our approaches we imagine for now that we have an infinite memory to hold all the running history of the software.

Taking this idea the question now is how to prioritize the headers in the queue that are waiting to be tried so that the number of tries (system calls) becomes minimum. We take three major prioritizing policies and compare them to see what difference they make.

1. Always try headers in the same order. (Naive approach)
2. Always try the header that has happened most in the first place then the header with second most occurrence and so on. (Hard priority)

\(^1\)An application that carries a higher layer application and we only have that higher layer header
\(^2\)In a uniformly distributed space of queries with the same probability that any of the above headers are absent (which is not so realistic) we still have a first try hit rate of almost 20\%
3. Try the header that has happened most with higher probability than others. (Soft Priority)

We examined these three approaches by generating queries with different sequence patterns of absent header. Let \( L \) be the number of layers. A packet could start with \( l^{th} \) layer which is \( 1 \leq l \leq L \). In our first example we generated packets that could start with Ethernet or IP or SCTP or Diameter – which is an application layer protocol. These packets could be of types \( 1 \leq l \leq 4 \). In our first input pattern we generated 100 input queries with uniformly distributed amounts of packets with absent headers\(^3\); Which means \( P_1 = P_2 = P_3 = P_4 = 0.25 \). Figure 5.1 depicts the cumulative distribution of the system calls for 100 input queries using three approaches.

In Soft Priority each time we pick a random header from the list according to the distribution \( P_s(L_n) \) and try it. if it is not the right header we remove the element from the list and redefine our distribution. \( L_n \) being the random variable assigned to our headers in \( n^{th} \) try we can define the distribution in each try according to the following formula,

\[
P_s(L_0) = N_L/N_T \\
P_s(L_n) = P_s(L_{n-1})/(1 - P_s(L_{n-1}))
\]

\( N_L \) is the number of occurrences of type \( L \) header and \( N_T \) is the total number of items in the history.

![Figure 5.1: Cumulative number of system calls for uniformly distributed input](image)

As can be seen in the figure in uniformly distributed case there is not much difference between these three approaches. We could expect this since in uniform case there are almost the same amount of packets of all kinds received in the history at any given time therefore our prioritizing policy makes no big change in the order that the headers are to be tried. However every now and then that the two or more consecutive queries happen to be the same type the slope of the diagram tends to decrease in case of soft and specially hard priority.

We repeat the experiment but this time our input query sequence is biased on one type. In the other words there is a 50% chance that the input is of type 2 at

---

\(^3\)We tried with a Diameter protocol packet and the input queries consisted of this packet with absent LinkLayer or absent Internet or absent Transport or a complete packet with no headers absent. (As the nature of the packet suggests we could not include a packet with absent application layer header but it makes no big difference in our conclusions of the study)
any given time. The result of the repeated experiment with 300 inputs for each approach is shown in Figure 5.2. The interesting phenomenon in this diagram is that in the beginning all three approaches are more or less the same. But at some point (around 150) Hard priority begins to show a better performance. The reason is that as the inputs’ history is accumulated in the memory the number of type 1 record becomes more and more significant comparing to other three types. In naive approach it makes no difference since we do not prioritize but in the hard approach the number grows farther and farther from the number of other input types and it is stuck to the top of the list and is always chosen first and good for us that in 50% of input queries it is actually the right choice causing the number of system calls to drop. The Soft priority approach is a little tricky. In this case we still prioritize according to the history and half of the buffer is populated with type 2 packets but still in every decision in case of type 2 inputs there is a 50% chance that we do not choose the type 2 header and this makes the system calls to grow.

In real world applications it is quite probable that for a long time the input remains the same type or equivalently one user might send a big amount of the queries of the same type. In the Figure 5.3 you can see the behaviour of the three approaches in regard to this phenomenon. Here in each approach we have fed the software with 100 queries of which the first 30 are of the same type and next 70 are of a different one. As can be seen from the figure in Naive approach the slope is the same since the input type is the same and after 30 the slope changes since the input type changes. In Hard priority approach the slope remains the same until 30 and is much less than Naive approach since in this case this type of header comes first with absolute priority. After 30 the input type changes but it is still the second priority in the list since there is still a record of 30 inputs of the previous type in the history. That explains the bigger slope right after 30. It goes on until there are 30 instances of the second type are submitted, which happens at iteration 60, now the number of the second type overcomes the number of the previous type and it goes to the top of the list and the slope decreases to what it was in the first place. In the Soft priority

\[\text{4One point is worth mentioning. As the diagram shows Soft priority and Naive approaches are more or less the same in terms of efficiency but this is not true. As in Naive approach we make a list and this list remains the same forever the more populated input type (in our case type 2) might be somewhere at the top of the list and this might cause the naive approach seem to have less or equal amount of system calls.}\]
approach the diagram shows a smoother path. Taking a closer look you can see at 30 the slope starts to increase but it is smoothed out and reaches the desired slope fast. However the overall amount of system calls is slightly more than Hard approach since we give less chance to the top priority packet to be actually tried first (like in Hard approach).

The difference in behaviour of Hard and Soft approaches is a very important remark. We illustrate the importance of this difference in Figure 5.4 in which we repeated the same experiment but in a longer run; With 350 input queries of which 150 are the same type and the rest are of a different type. As you can see in the figure the Soft approach outperforms the Hard approach when catching up to replace the top priority in its list at iteration 250. However later on Hard priority catches up near 350 and becomes better again.

This behaviour illustrates an example of an undesirable phenomenon in which when there are a lot of requests for one type its record outnumbers other types and it becomes stuck at the top of the list while there is no more request for it. This is a fundamental problem with Hard priority approach. In case of soft priority approach this problem is smoothed out by providing a chance to other types every now and then to become top in the list. Taking this approach the top priority record is more reluctant to becoming top in the priority list and therefore its count becoming significantly bigger than other types in the history.
In fact the problem becomes even more severe when considering irregular input sequences due to frauds or mistakes. In such cases the software can be made to accommodate very large and different records. In those cases the items in priority list are fixed in their places in Hard approach and our algorithm can do no better than in Naive approach.

This problem can be viewed as being the result of keeping a too long history also. In fact the history expires after a while and we should only take into consideration the relevant and new data to decide on the type of next input. Furthermore, recall that we considered to have an infinite buffer to keep the record of all previously submitted enquiries. Well, this is not the case in reality and we have to consider a buffer size to keep some limited amount of history to use in our algorithm. Considering the problem with Hard priority as being a result of keeping too much history, we can define a buffer size. A very short buffer — e.g. to decide only on the last result — would lead to ignoring a large amount of useful information and it becomes pretty much like the case with no buffer but resolves the problem with Hard priority. A very large buffer provides a more insightful decision on the next type to be expected but again suggests the problem with Hard priority.

From all the discussions before we should take into consideration two very important criteria that affect the performance of our system,

- Fast response to highly frequent changes in input pattern. Meaning that it shouldn’t take so long for the software to decide if the input type has changed thus it should come higher in priority list.

- Fast response to less frequent changes in input pattern. Meaning that despite having a long history of one type — say A — in case of receiving type B the software should be able to converge fast into reordering the priority list so that type B comes up in place of type A.

In order to elaborate more on this issue we should consider another important factor as well. These diagrams so far show the overall performance of our system but what we really are interested in is the user experience of the response time of the system. The four diagrams in Figure 5.5 show the performance of our system in terms of the user satisfaction and cumulative number of system calls. These are two experiments with 350 inputs for which a history window size of 300 is used. In Figure 5.5(a) you can see the system’s performance with an input pattern of 50% of type 2 packets dispersed among other types. In Figure 5.5(b) you can see the system’s performance with an input pattern of 150 type A packets in a row followed by 200 of type B. Using these two diagrams we will try to investigate the performance of Soft and Hard priority under two aforementioned criteria.

As is shown in the figures, Hard priority works better in highly changing traffic; Since it can assign priorities so fast and replace the higher with the one below. But in Soft priority approach the real higher priority type gets only 50% chance of being chosen first. However in the second experiment Soft shows a better performance since it did not allow the higher priority record become so big so that it takes a long time for the new packet to replace it on top of the list; What happened actually to the Hard case.

\footnote{We have eliminated the diagram for Naive approach since its scales were relatively bigger and made the other two diagrams’ difference appear vague.}
Furthermore from Figure 5.5 you can see a measure of user satisfaction. This diagram shows the distribution of the number of system calls during each experiment. In other words you can see that for example in the first experiment around 190 users (out of 350) experienced a delay equal to 2 system calls in the Hard approach comparing it to around 100 users in the Soft approach or 50 in the Naive approach. You will see later a summary of these information in table 5.1 and 5.2.

The above observations bring us to a third approach which is a compromise between Hard and Soft approaches. We need a dynamic algorithm that Hard prioritize the list in case of highly fluctuating traffic and Soft prioritize in case of a large number of requests for the same type of input. In order to reach for such a method we need to consider our prioritizing approaches from a different point of view. When we have a fluctuating traffic with a biased probability of let’s say type 1 traffic be 50% we are actually expecting 50% of our buffer to be populated by type 1 packets at any given run of the program. It means that if for any reason the traffic pattern changes and becomes biased to another type we have to fill the buffer as fast as possible so that the new type replaces the old one. For our example of 50% this process takes effect relatively fast as you can see from Figure 5.5(a). Now what we can do is to let the buffer fill up to 50% in a Hard Prioritizing manner then if the top item in the list exceeds 50% of the buffer we change the plan and order our buffer in a Soft prioritizing manner. This way we ensure that for highly fluctuating traffic we will give the absolute chance for the higher item in the list to be chosen but if it starts to overpopulate our buffer we take the Soft approach in order to prevent it from getting stuck at the top of the list. We propose 50% since in
real world application it is quite probable that up to 50% of the input queries be the same type. But if it exceeds 50% it can be an indication of abnormal behaviour in input submissions. In other words our probability distribution for each invocation is as the following,

\[
P_d(L) = \begin{cases} 
P_s(L), & \text{if the most populated header in the buffer occupies more than 50% of the buffer} \\
1, & \text{if } l \text{ is the most populated header in the buffer and it occupies less than 50% of the buffer} \\
0, & \text{if } l \text{ is not the most populated header in the buffer and the most populated header in the buffer occupies more than 50% of the buffer} 
\end{cases}
\] (5.3)

We implemented our Dynamic approach and compared it to the other two approaches in the same two before mentioned cases. In Figure 5.6 you can see the results of our Dynamic approach. As can be seen, our Dynamic approach

\begin{figure}[h]
\centering
\subfloat[50% biased input (fast changing traffic pattern)]{
\includegraphics[width=0.45\textwidth]{figure1}
}
\subfloat[packets of the same type (slow changing traffic pattern)]{
\includegraphics[width=0.45\textwidth]{figure2}
}
\caption{Comparison of the performance of Hard and Soft and Dynamic priority in response to slow and fast changing input pattern}
\end{figure}

in Figure 5.6(a) tries to imitate the shape of Hard priority approach which is to Hard prioritize packets since there is most of the time less than 50% of the buffer populated with the same type packets and in Figure 5.6(b) the Dynamic approach tries to imitate the Soft priority approach which is more efficient than Hard in those circumstances.

In terms of user satisfaction as is demonstrated in table 5.1 in the case that the input is biased on one type 180 users have experienced 4 system calls in Naive approach. In other three approaches most users experienced 2 system
calls but, comparing hard case with 174 users and soft case with 102 users experiencing 2 system calls we can see a big difference in user experience while in Dynamic case the user experience is much better than in Soft case and with 154 users experiencing 2 system calls is quite close to Hard case.

<table>
<thead>
<tr>
<th>Number of System Calls</th>
<th>Naive</th>
<th>Hard</th>
<th>Soft</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>52</td>
<td>174</td>
<td>102</td>
<td>154</td>
</tr>
<tr>
<td>4</td>
<td>180</td>
<td>59</td>
<td>99</td>
<td>73</td>
</tr>
<tr>
<td>6</td>
<td>63</td>
<td>55</td>
<td>76</td>
<td>63</td>
</tr>
<tr>
<td>8</td>
<td>55</td>
<td>62</td>
<td>73</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 5.1: Comparison of user experience in the biased input type

In the second experiment with input pattern of the same type header repeated a large number of times, The user experience in Naive case is so bad with no user experiencing any system calls smaller than 6. However in Hard case with 199 users with 2 system calls and 149 with 4 and comparing it to the Soft case with 221 users with 2 system calls and 123 with 4 we can see that soft is working better in this case while Dynamic approach again leans toward the better performance with 208 users experiencing 2 system calls and 131 experiencing 4.

<table>
<thead>
<tr>
<th>Number of System Calls</th>
<th>Naive</th>
<th>Hard</th>
<th>Soft</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>199</td>
<td>221</td>
<td>208</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>149</td>
<td>123</td>
<td>131</td>
</tr>
<tr>
<td>6</td>
<td>200</td>
<td>2</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>150</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5.2: Comparison of user experience in the same type input

In order to make an overall evaluation of our method we designed a more realistic scenario of the input sequences. Our scenario consists of 1050 enquiries for each of the 4 approaches. 350 first inputs are uniformly distributed inputs of all types. Next 150 would be of one type only and afterwards the next 200 inputs of a different type. Finally the last 350 inputs are biased inputs with 50% chance of one type being the input type. Here in Figure 5.7 you can see the cumulative distribution of the number of system calls along with the user satisfaction diagram. From Figure 5.7 the cumulative system calls the adopting

Figure 5.7: Realistic Scenario
behaviour of our Dynamic method can be clearly seen that in each stage of the
input pattern it tries to imitate the best approach. Table 5.3 illustrates the
number of relatively satisfied users with a experience of 2 system calls are big
equal in Dynamic method with respect to the number of unhappy users with
the number of 6 or 8 system calls and these numbers are smaller than those of
Soft approach. However if you take a snapshot of the runs of the program at
any given time you can see that the Dynamic method is closer to the better
approach.

There is another important thing to notice in the figures. It is true that in
the Hard approach there are more users that experience 2 system calls but in
general our system might operate with 2 or 4 or 6 or 8 system calls. waiting
time for 2 and 4 system calls does not differ that much comparing them to waiting
time for 6 or 8. Therefore if we pay the price of having less users with 2 system
calls and more with 4 system calls in turn we gain the advantage of having the
least users with 6 or 8 system calls. In other words we prefer the diagram of
user satisfaction to be a smooth and straight line rather than having a peak at
2 system calls but having also maximums at 6 or 8 which are not desirable user
experiences at all.

<table>
<thead>
<tr>
<th>Number of System Calls</th>
<th>Naive</th>
<th>Hard</th>
<th>Soft</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>153</td>
<td>445</td>
<td>324</td>
<td>381</td>
</tr>
<tr>
<td>4</td>
<td>258</td>
<td>315</td>
<td>316</td>
<td>269</td>
</tr>
<tr>
<td>6</td>
<td>350</td>
<td>146</td>
<td>233</td>
<td>222</td>
</tr>
<tr>
<td>8</td>
<td>289</td>
<td>144</td>
<td>177</td>
<td>178</td>
</tr>
</tbody>
</table>

Table 5.3: Comparison of user experience in the same type input
Chapter 6

Conclusion

Generally the flow of understanding a network packet as is done in real world is divided into some major stages; Sniffing the bits from wire and storing them onto the memory in a proper format (such as hex numbers), then retrieving the hex numbers from memory and converting them into proper formats – such as adding additional information like timestamps, etc... – in order to be understandable by packet analyzer tools, and finally comes the packet analyzer that provides the useful information about the packet in a user friendly format. The Online Packet Analyzer (OPA) developed in this thesis is an online service to facilitate the task of packet analysis merging the two last stages. There are serious considerations when trying to automate a human task; These hex streams of meaningful packets could be embedded inside a lot of other irrelevant text that sometimes very much look like meaningful packets. These packets can be extracts of headers rather than a complete packet. Here in our work we tried to find a solution for these problems by guessing the missing parts of the packet and trying them one after another to see which one makes a meaningful and dissectable packet. This type of adding headers is generally done by engineers that have a former view of the proper header to add. These kinds of knowledge are very hard to be embedded inside a computer code. However our prioritizing policies that were discussed in the last chapter was an attempt to get closer to human efficiency in guessing the absent headers.

The database that comes along with the OPA service is also intended to facilitate the troubleshooting of the network. The ability to update this database by the experts in the field provides an opportunity to have a valuable collection of side material that can be beneficial for the support experts as well as the newly started interns that want to learn new material.

This service is made specifically to be employed inside Ericsson support but such an online service could be beneficial everywhere. This is a light weight web page therefore one can even use it from his/her mobile phone. This means to have a Wireshark almost anywhere.

The OPA can be used for educational purposes. Imagine finding any example of packet data protocols in the website with all the educational side material and links to manuals and white papers and standards all in one page. Also considering the feature of sending test packets, a webpage can be imagined with
all the necessary building blocks of a packet. One can easily choose header
types and build his/her own packet and send it to a destination and observe the
packets returned and actually see what is happening. Such a feature not only
makes it easy for the support guru to test their systems but also is of a great
educational value. The students who just have learnt network protocols can put
what they learn to practice. They will never forget it this way!

For the future improvements this service can be considered as a script that
processes any text document from the html pages in a browser to any text file
on the local hard drive. The result will be a document with all the hex byte
stream like text inside it highlighted. Then it only takes a click to find out what
packet the stream represents.

The efficiency of the header guessing method can also be subject to im-
provement by considering an algorithm that finds a more precise probability
distribution by looking into the history then predicting the next packet to be
expected from that distribution.
Bibliography


[7] [https://sites.google.com/site/lteencyclopedia/home](https://sites.google.com/site/lteencyclopedia/home), Last Update 22 Feb. 2012.


