THE REPUBLIC OF TURKEY
BAHCESEHIR UNIVERSITY

BmtGen: A TRAFFIC GENERATOR FOR TESTING DPI TOOLS

Master Thesis

KÜRŞAT KOBYA

ISTANBUL, 2016

THE REPUBLIC OF TURKEY
BAHCESEHIR UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
COMPUTER ENGINEERING

BmtGen: A TRAFFIC GENERATOR FOR TESTING DPI TOOLS

Master Thesis

KÜRŞAT KOBYA

Supervisor: ASST. PROF. DR. YÜCEL BATU SALMAN

ISTANBUL, 2016

THE REPUBLIC OF TURKEY
BAHCESEHIR UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
COMPUTER ENGINEERING

Name of the thesis: BMTGEN: A TRAFFIC GENERATOR FOR TESTING DPI TOOLS
Name/Last Name of the Student: Kürşat KOBYA
Date of the Defense Thesis: 27.05.2016

The thesis has been approved by the Graduate School of Natural and Applied Sciences.

Assoc.Prof. Nafiz ARICA
Graduate School Director

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Science.

Asst.Prof.Dr. Tarkan AYDIN
Program Coordinator

This is to certify that we have read this thesis and that we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Committee Members	Signature
Thesis Supervisor Asst.Prof.Dr. Yücel Batu SALMAN	----------------------------------
Member Asst.Prof.Dr. Tarkan AYDIN	----------------------------------
Member Asst.Prof.Dr. S. Serkan Güllüoğlu	----------------------------------

ABSTRACT

BMTGEN: A TRAFFIC GENERATOR FOR TESTING DPI TOOLS

Kürşat Kobya

Computer Engineering

Thesis Supervisor: Asst.Prof.Dr. Yücel Batu Salman

May 2016, 38 pages

The increasing personal usage of new applications of LTE networks motivates operators to research in-depth traffic profile. In this respect, operators started to develop deep packet inspection applications for optimizing their networks performance and improving network utilization. To be able to test aforesaid, DPI applications a necessity arise. Traffic generators are needed to test and investigate performance of DPI applications.

There are several hardware and software based traffic generators made for similar purpose, yet all of them have different drawback and discommodity to test DPI utility runs on LTE networks.

The purpose of this study is to offer a solution to meet indicated demands. In this context, implementation of a new traffic generator algorithm that support Napatech Gigabit Ethernet Cards and produce meaningful near real time traffic is investigated and explained detail.

Keywords: Traffic Generators, DPI

ÖZET

BMTGEN: DERİN PAKET İNCELEME PROGRAMLARI İÇİN TRAFİK ÜRETİCİ TASARIMI VE UYGULAMASI

Kürşat Kobya

Bilgisayar Mühendisliği

Tez Danışmanı: Yrd. Doç. Dr. Yücel Batu Salman

Mayıs 2016, 38 sayfa

LTE ağlarda yeni uygulamaların bireysel kullanımındaki artışı, operatörleri ağ akış profillerini derinlikli incelemeye zorladı. Bu kapsamda operatörler, ağ performanslarını geliştirmek ve ağ kullanımlarını arttırmak amacı ile derin paket inceleme uygulamaları geliştirmeye başladı. Bahsi geçen uygulamaların geliştirilmesi ile bu uygulamaları test etme ve performanslarını inceleme ihtiyacı ortaya çıktı. İşte bu açığı karşılamak için trafik üretici uygulamalara gereksinim duyuldu.

Benzer amaçlar için geliştirilmiş çeşitli donanım ve yazılım tabanlı trafik üretici uygulamalar var ise de, bunların her birinin farklı bir eksikliği veya LTE ağlarda derin paket inceleme yazılımlarına uygunsuzluğu bulunmakta.

Bu çalışmanın amacı yukarıda bahsedilen ihtiyaçları gidermeye yönelik bir çözüm sunabilmektir. Bu bağlamda, Napatech Gigabit Ethernet Kartlarını kullanarak gerçek zamanlı trafik üretebilen bir algoritmanın gerçeklenmesi işlemi incelendi ve detaylıca anlatıldı.

Anahtar Kelimeler: Traffic Generators, DPI

TABLE OF CONTENTS

TABLES viii

FIGURES ix

ABBREVIATIONS x

1. INTRODUCTION 1

1.1 DEEP PACKET INSPECTION 2

1.2 P-GW in LTE NETWORKS 3

1.3 S-GW in LTE NETWORKS 3

1.4 MME in LTE NETWORKS 3

1.5 ROLE OF DPI BETWEEN MENTIONED NODES 5

1.6 PREVIOUS WORK 5

1.7 THESIS ROADMAP 8

2. DESIGN AND IMPLEMENTATION 9

2.1 PROGRAMMING LANGUAGE AND COMPILE OPTIONS 9

2.2 DIRECTORY STRUCTURE 10

2.3 PROJECT FILES 11

2.4 DATA MODEL AND STRUCTURES 12

2.5 PROGRAM FLOW 13

2.6 USED LIBRARIES 14

2.7 IMPLIMENTED AND USED FUNCTIONS 16

2.8 COMMAND LINE ARGUMENTS AND USAGE 21

3. ENVIRONMENT AND TESTS 24

3.1 ENVIRONMENT 24

3.2 VERIFICATION TEST 25

3.2.1 Verification Test Results 26

3.3 LOAD TEST 26

3.4 FEAUTURE PERFORMANCE TESTS 27

3.4.1 Only Ip Address 28

3.4.2 Ip Address And Port 28

3.4.3 Only Gtp Address 28

3.4.4 Gtp Address And Port 29

3.4.5 Ip Address And Gtp Address 29
3.4.6 Ip Address, Gtp Address And Port 29
3.4.7 Ip Address And Ip Checksum 30
3.4.8 Ip Address, Port, Ip And Port Checksum 30
3.4.9 Ip Address, Gtp Address, Port, Ip And Port Checksum 31
3.4.10 Features Performance Test Results 31
3.5 MULTI PROCESS TEST 32
3.5.1 Multi-process Test Results 33
4. DISCUSSION 34
5. CONCLUSION 35
BIBLIOGRAPHY 36

TABLES

Table 3.1: Feature test results...31

FIGURES

Figure 1.1: LTE Network Architecture 4
Figure 2.1: Project Directories……. 11
Figure 2.2: Top Level Design……. 14
Figure 2.3: Napatech API Architecture……... 16
Figure 3.1: Output of profiling tool…….. 26

ABBREVIATIONS

API	:	Application Programming Interface
APN	:	Access Point Name
CPU	:	Central Processing Unit
DPDK	:	Data Plane Development Kit
DPI	:	Deep Packet Inspection
EPC	:	Evolved Packet Core
FPGA	:	Field Programmable Gate Array
GCC	:	The GNU Compiler Collection
GTP	:	GPRS Tunneling Protocol
GTP-C	:	Gprs Tunnelling Protocol Control Part
GTP-U	:	Gprs Tunnelling Protocol User Part
GUI	:	Graphical User Interface
IDS	:	Intrusion Detection System
IPS	:	Intrusion Prevention System
LTE	:	Long-Term Evolution
MME	:	Mobility Management Entity
NIC	:	Network Interface Card
OSI	:	Open Systems Interconnection
P2P	:	peer-to-peer
PCAP	:	Packet Capture
PDN	:	Packet Delivery Network
P-GW	:	PDN Gateway
QoS	:	Quality of Service
S-GW	:	Serving Gateway
UDP	:	User Datagram Protocol

1. INTRODUCTION

The increasing personal usage of newly created implementations related computer networks in every aspect; consistently motivate operators to research for the invention of new network equipment in aim of understanding of in-depth traffic profile. In this respect, operators started to develop deep packet inspection applications for optimizing their networks performance and improving network utilization. Deep Packet Inspection (DPI) is way of classing incoming traffic description by the method of investigating packet payload details, instead of just studying at the specified data found in headers of packet. Packet payloads are investigating to be able to match across a signature database which contains different protocols unique expressions [2]. To be able to test those applications a necessity arise. To test and observe performance of DPI applications, particular protocols, or any desired packet inspection conditions, traffic generators are needed.

The packets shall be generated for certain traffic patterns according to an occasional specification of user criteria based on a beforehand picked pcap files. Manufacturers of network equipment utilize traffic generators to examine their product in the laboratory setting and to exhibit their competences to their customers. Benchmark examinations are executed in assessment laboratories to examine and ratify the equipment from varied producers by the help of very skillful and costly traffic generators [3].

On the other hand, there are several traffic generators made for replay given pcap to particular port and IP addresses. In this purpose, network acceleration tools and network interface cards (NIC) build their own traffic generators. Napatech traffic generator for Napatech NIC’s, Intel’s data plane development kit (DPDK) packet_gen for DPDK based NIC’s are some of software based traffic generators to work on specific network cards. Also there are many open source traffic generator project like tcpdump, ostinato, solarwinds and many others [17]. Yet many of these solutions based on an algorithm to take all frames of packet captures to heap and sends from memory to destination. DPI tools are designed for huge amount of real time or near real time traffic; hence to provide what mentioned tools await, pcaps should be very big captures. This behavior of algorithm let the generator programs allocate huge amount of memory and as a result it is inevitable to encounter stack overflow and segmentation faults on system.

To deliver expected packet stream behavior, we propose to implement a new traffic generate algorithm that support Napatech Gigabit Ethernet Cards and produce meaningful near real time traffic. In this algorithm only small piece of a real time traffic capture will be used as a template and by manipulating requested header of captured packet, many (infinite) different flow will be produced to serve DPI tools. We will mention details and design of proposed traffic engine later.

The essential aim of this traffic generator is to supply realistic data for DPI which is designed for mainly the packet data network gateway (P-GW), the serving gateway (S-GW) and mobility management entity (MME) nodes of LTE Networks. Thus it is found beneficial to explain following items briefly.

1.1 DEEP PACKET INSPECTION

Deep packet inspection (DPI) is a nubile method of filtering packets that performs at the Application layer of the Open Systems Interconnection (OSI) reference model. The utilization of DPI let it possible to identify, find, categorize, redirect or block packets with exclusive information or code payloads that traditional packet filtering, which tests only packet headers, are enable to detect [7].

Utilizing DPI, communications service providers can assign convenient resources to regulate traffic flow. For instance, a communication packet tagged as high priority can be directed to its destination in front of lesser important or low-precedence messages or messages involved in ordinary browsing of Internet. DPI can also be utilized for choked information transfer to prevent peer-to-peer (P2P) abuse, enhancing network throughput for most subscribers. Security related implications of DPI are extensive because the technology renders it possible to describe the inventor or receiver of context containing exclusive packets, an ability that has create concern throught advocates of online privacy.

1.2 P-GW in LTE NETWORKS

The packet data network gateway is the Evolved Packet Core (EPC)’s exit gate with the world outside. Through the SGi interface, every PDN gateway interchanges information with one or more exterior devices or packet data networks, like servers of the network operator, the internet or the IP multimedia infrastructure. In other words, The P-GW is in charge of allocation of IP addresses for the user equipment (UE). Every packet data network is defined by an access point name (APN) [1]. Typically, a network operator utilizes lots of different APNs, for instance one for the internet and one for its own servers [26]. Each UE is appointed to a default PDN gateway when it first opens, to provide it always-on connectivity to a default packet data network such as the internet. Figure 1.1 shows P-GW position in LTE Network.

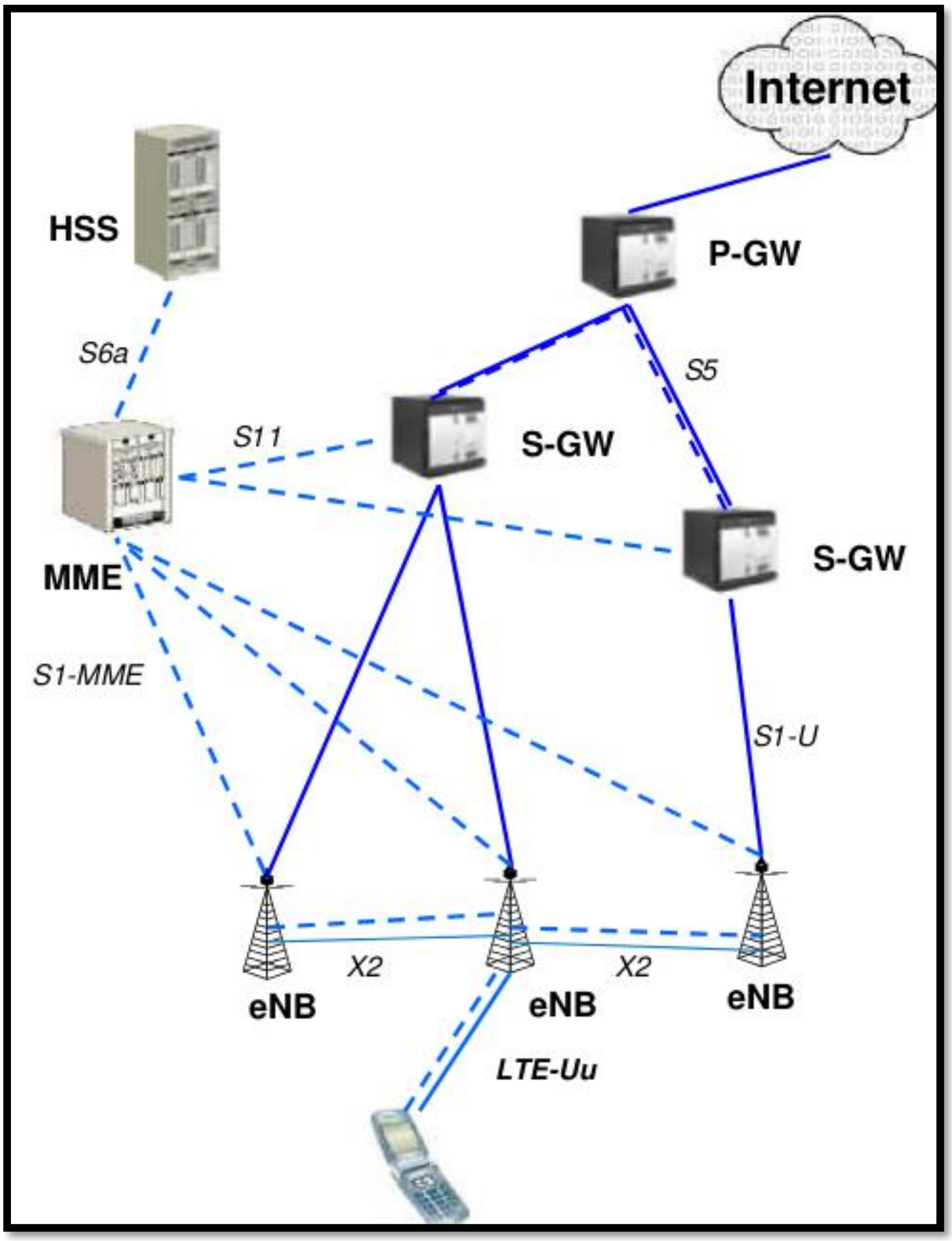
1.3 S-GW in LTE NETWORKS

Every IP packets of user are wandered between nodes using the S-GW. The serving gateway acts as a router, and routes data between the base station and the PDN gateway. An LTE network might typically contain many serving gateways, every one of which takes care and looks after of the UE’s in a particular geographical area. Every UE is assigned to a one serving gateway, but the serving gateway can be altered if the UE change its location sufficiently far to join district of other S-GW [26]. Figure 1.1 shows S-GW position in LTE Network.

1.4 MME in LTE NETWORKS

The mobility management entity is a node to control the high-level operations of the mobiles, by transferring its control plane messages about issues such as security and the management of line of packets that are not related to radio access communications. As it was with the serving gateway, a network might typically contain lots of MME's, each of them is responsible for a particular geographical area. Each mobile is assigned to a single MME, which is also known as its serving MME [31]. Figure 1.1 shows MME position in LTE Network.

Figure 1.1: LTE Network Architecture



1.5 ROLE OF DPI BETWEEN MENTIONED NODES

Using specified DPI for particular network nodes mandates the DPI process to decode specific protocols needed per node. For instance; in MME node, gtp-c protocol should be decoded whilst S-GW node dealt with gtp-u protocols. To answer these expectation DPI modules should be programmed due to its environment. In this manner, proposed new traffic generator algorithm must support tcp, udp and gtp protocols.

1.6 PREVIOUS WORK

In academic literature, Traffic Generators often designed on hardware platforms. And this hardware platform is commonly FPGA (Field Programmable Gate Array). Depending on the fact that today's FPGA have very big amount of logic gates, it make s it possible to creation of complex hardware platforms and also FPGA based systems are not create burden for CPU performance [13]. However, there are some drawbacks to use hardware platforms for packet generators. One problem is this kind of generators requires technical competences that restrict the scalability and flexibility of the design [3]. Also using on-board processors and extra hardware necessity makes this kind of generators useless for LTE specific DPI's [9]. Hence the proposed solution has to be based on pure software.

There are also numerous different software based traffic generators were proposed. Lots of all too common solutions that are frequently discussed in papers are used for mostly artificial traffic generation. These traffic producers are mostly served for stress testing servers and firewalls productivity examination.

The most prevalently utilized traffic generator is Iperf, [5-11-12] which produces packets with UDP protocol at a given rate or packets with TCP protocol at highest possible throughput. Iperf is commonly utilized for computation of the highest possible bandwidth on IP networks. It makes it possible of adjusting of several parameters relevant to protocols, timing and transmission buffers. For each examination it reports the bandwidth, packet loss, and other parameters. However, Iperf has performance issues when it comes to replay User Datagram Protocol (UDP) packages above ten gigabit per second.

BRUTE [23] was also introduced as another application claiming itself to increase the accuracy of the output speed rate. BRUTE that is abbreviation of “Brawny and Robust Traffic Engine”, address the problem of reliable device testing, by using advance power of the Linux kernel in order to create accurately produced traffic flows up to high amount of bitrates.

Other proposals, like TG [29] or MGEN [21] mainly designed for calculate statistical information about the Packet Sizes (PS) and Inter Packet Times (IPT) information. TG is a traffic generation application that produces only one-way UDP or TCP packets of lines. The traffic is described in terms of arrival times and packet lengths. The Multi-Generator (MGEN) is another software solution made by the Naval Research Laboratory which supplies the ability to provide IP network throughput examinations and computations using UDP and IP. The set of tools generates real-time traffic behaviors so that the network can be loaded in many of ways.

Furthermore, Ostinato [16] is another generator presented recently, written with python, where users may arrange streams with different features. Ostinato has agent-controller architecture. There are two corresponding application, one is graphical user interface (GUI) agent and the other one is the controller. The GUI agent runs on a computer and connects to one or more computers running the controller. The GUI agent can then access and control all ports of all connected controllers. The controller can be connected and shared by multiple GUI agents at the same time.

Since all these software based traffic generators create packets with synthetic or dummy payload; it is not possible to use them for specific DPI testing purposes. This behavior is not well for testing DPI tools, since aforesaid tools are installed for gathering statistically particular performance indicators; they need specific packets as an input, which well-known traffic generator are lack of producing so.

Traffic generators which algorithm depends on replay, aim to inject previously collected packet captures to the network. The most reputed product made for this aim is Tcpreply [28] that is a user-space program for replying pre captured pcap files onto the network. Tcpreplay presents to user the ability to take captured traffic in pcap format as an input to examine many network devices [30]. Tcpreplay is used by lots of intrusion detection system (IDS), firewalls, intrusion prevention system (IPS) and other networking equipments.

TCPivo[32] is a kernel-space tool to mirror network traffic which has purpose to develop the precision of the packets timing. TCPivo basically read package from file, arrange transmission rate as given argument, and send package to given port. Within the context of functional design TCPivo work exactly as Tcpreplay.

There are also more complicated traffic producers which are able to imitate the process of beforehand recorded network traces by more sophisticated modeling of traffic. Harpoon [14] is an example this kind of generators. It is generator base on network flows that imitates netflow calculations by investigating various flow statistics. Harpoon is implemented for creating packet traffic that has the same payload and header as traffic observed at live servers. Traffic generation with Harpoon uses either UDP or TCP for transportation.

Swing [16] is a flexible packet generator that is able to take data for network conduction of real computations. Swing is network responsive traffic generator that starts from controlled traffic at a particular point in the network, and then automatically takes distributions for user. It then produces live traffic packets related to the corresponding models in a network environment. Swing also permits the consumer to change hypothesis about network situations, implementation mixture and implementation differential features, e.g. response size of equipment, to generate new traffic.

Tmix [18] is another tool based on characterization of TCP connections that extract an input pcap trace grabbed from a given link of a network place.

Another solution to provide traffic for DPI tools is presented lately by Cisco Company. TRex is an open source, stateful traffic generator runs on DPDK. It generates replay of real traffic templates and amplifies both client and server side traffic. On the other hand it depends on DPDK based network interface cards and not compatible with Linux Networking Stack.

The joint deficiency of these mentioned generators is that the calculations includes user sensitive data and also these solutions based on an algorithm to take all frames of packet captures to heap and sends from memory to destination. DPI tools are designed for huge amount of real time or near real time traffic; hence to provide what aforesaid tools await, packet captures should be very big captures. This behavior of algorithm let the generator programs allocate huge amount of memory and as a result it is inevitable to encounter stack overflow and segmentation faults on system. Also these approaches cannot provide realistic packet payloads if they tried to spend lesser heap memory.

1.7 THESIS ROADMAP

Remainder of this thesis will contain, design decisions, plans, options and top level design of proposed traffic generator; brief explanation of used libraries for implementation; brief explanation of test environment; test results and summary.

2. DESIGN AND IMPLEMENTATION

Our aim to develop and implement a brand new traffic generator based on aforesaid requirements. To meet demands, it is planned to develop a software traffic generator that basically reads pre captured small pcap files, manipulates internet headers, protocol headers, checksums and push newly generated network frame to output buffer. The proposed packet generator, BmtGen should endure change of related fields, provide different network flows every time, should be able to run multi-process, should be able to send pcap on high speed NAPA network cards.

2.1 PROGRAMMING LANGUAGE AND COMPILE OPTIONS

BmtGen was decided to be coded with C programming language. C programming language is a general-purpose, imperative computer programming language, promoting structural programming. C programming language has been first developed by Dennis Ritchie starting from1969 till 1973 at Bell Labs, and used to implement operating systems later. Main reasons behind selection of programming language can be listed as:

a. When it comes to performance C programming language is unbeatable [19]. Since we are trying to create high performance software based traffic generator that works on high performance network interface cards, performance of language is important.
b. The operating system that DPI tool runs on it is Linux and UNIX. Major parts of these operating systems are coded in C. Thus in future if any software enhancement needed about socket adjustment, it would be much easier to correspond with c language.
c. Network Interface Card’s drivers are coded in C. Coding BmtGen with c programming language is going to give us an opportunity to use and alter driver related codes when necessary.
d. C programming language provides programmer to access to the many elements of computer. It opens a way to direct access to memory of CPU. This feature is very beneficial especially when we want to reach sockets buffer.

e. The libraries we have planned to use are coded in C and they have their own API for c programming language.

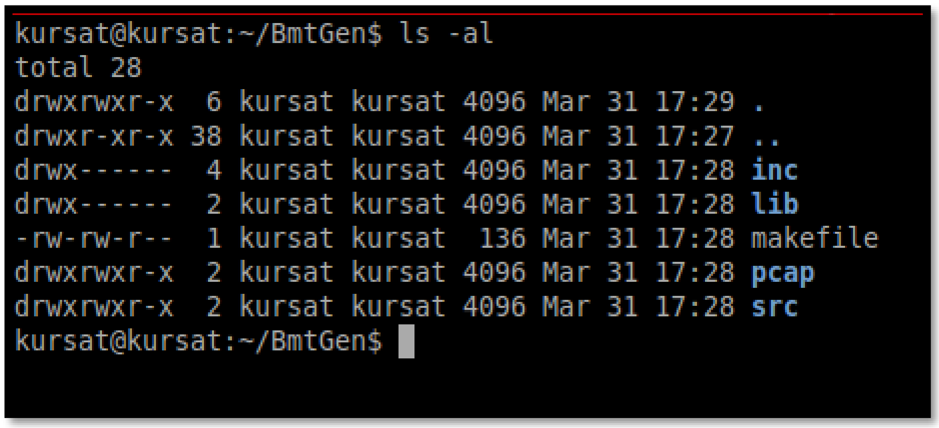
GCC compiler is used to compile BmtGen project. The GNU Project has produced the GNU Compiler Collection (GCC) as a compiler system [10]. GCC is a very important constituent of the GNU tool chain. The C programming language is handled by originally named the GNU C Compiler. GCC has been ported to many processor architectures, and is available for most embedded platforms. GCC has been accepted as the standard compiler by lots of different modern Unix-like computer operating systems, including Linux and the BSD family. Since it is the default compiler choice for most Unix-type systems and behavior of highly portable structure, GCC is best option in BmtGen project.

As version control system git is chosen. Git is a version control system that is mostly used for development of software projects. It is a distributed revision control system with an care about speed and data integrity [25]. Linux kernel developers designed and developed Git in 2005. Git presents many advantages like, being fast, letting programmers to work offline and making undo is possible almost for every cases.

2.2 DIRECTORY STRUCTURE

Directory structure in BmtGen project is decided as follows. In main directory, four directories are presented. “inc” directory stands for encapsulating header files, while “src” directory covers source files. “lib” directory contains external libraries that BmtGen application uses and “pcap” directory corresponds to pre captured template files that user should provide.

Figure 2.1 Project Directories


```
kursat@kursat:~/BmtGen$ ls -al
total 28
drwxrwxr-x  6 kursat kursat 4096 Mar 31 17:29 .
drwxr-xr-x 38 kursat kursat 4096 Mar 31 17:27 ..
drwx------  4 kursat kursat 4096 Mar 31 17:28 inc
drwx------  2 kursat kursat 4096 Mar 31 17:28 lib
-rw-rw-r--  1 kursat kursat  136 Mar 31 17:28 makefile
drwxrwxr-x  2 kursat kursat 4096 Mar 31 17:28 pcap
drwxrwxr-x  2 kursat kursat 4096 Mar 31 17:28 src
kursat@kursat:~/BmtGen$ 
```

2.3 PROJECT FILES

“inc” folder involves two folders and one header file.

a. pcap_related.h : this header file contains structures for saving template network frames in heap.
b. ntapi : this folder contains sixteen header files which are used by ntapi library. ntapi is the library that let us use Napatech network interface card’s drivers and socket buffers.
c. pcap : this folder contains eight header files which are used by pcap library. Pcap library is used to read template packet captures from file.

“src” folder involves two files.

a. pcap_related.c : this file is used to write methods that captures frames from pre-stored input packet capture file.
b. Traffic_gen.c : this file is used to write methods that manipulate network frames and sending them over Napatech NIC’s.

Lib folder contains pcap and napatech api libraries.

2.4 DATA MODEL AND STRUCTURES

To meet requirements, two main structures are needed. One for configuring the generator and the other was to keep template packet capture frames.

The “conf” structure, stores parameters about configuration of running process, like; start/end ip addresses to declare border for manipulating ip addresses, start/end port to declare border for manipulating ports, start/end inner ip addresses for gtp ip address range, toggle bytes for enabling/disabling options, bitrate, transmit port.

```
typedef struct _conf {
        uint32_t start_ip;
        uint32_t end_ip;
        uint16_t start_port;
        uint16_t end_port;
        uint8_t ip_toggle;
        uint8_t port_toggle;
        uint8_t ip_checksum_toggle;
        uint8_t protocol_checksum_toggle;
        uint32_t gtp_start_ip;
        uint32_t gtp_end_ip;
        uint8_t gtpu_toggle;
        uint32_t bitrate;
        uint16_t tx_port;
} conf;
```

The pcaps structure, stores every frame in the template pcap file, byte count of every frame, virtual lan information of every frame and structural stats.

```
struct _pcaps {
        unsigned char **item;
        unsigned int *byte_count;
        unsigned char *is_vlan;
        int item_count;
        int alloc_chunk;
        int biggest_frame;
};
```

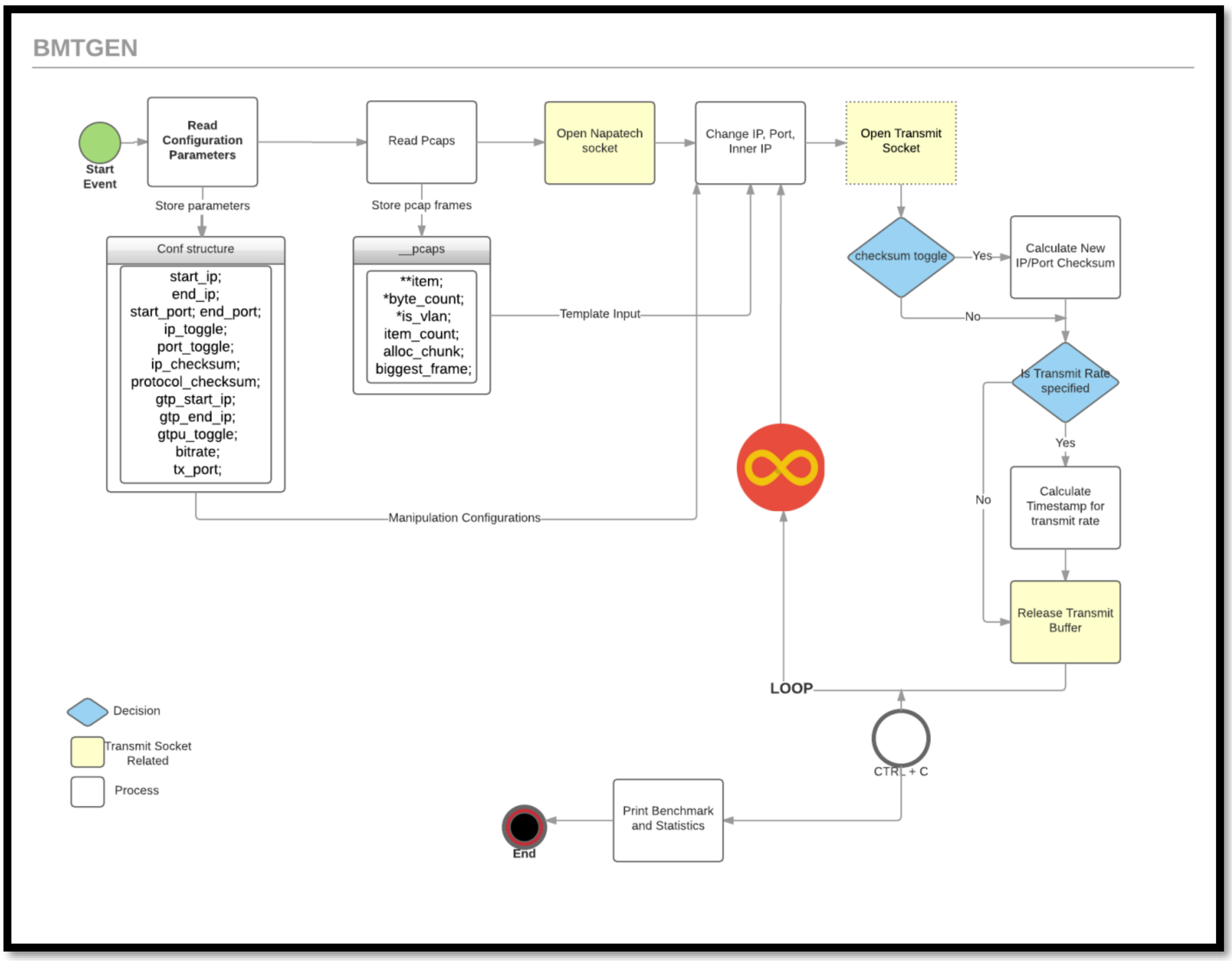
2.5 PROGRAM FLOW

The flow of proposed BmtGen and its top level design as follows.

a. The generator starts by reading configuration parameters from command line. Then it parses command line options and stores related parameters into conf structure.
b. Thereafter BmtGen starts to read pre captured packet from pcap folder and save all network template frames to heap and stores all information in _pcaps structure.
c. Then Napatech socket is initialized and opened.
d. Then generation phase starts, due to configurations and options, new ip addresses and ports are calculated.
e. Manipulated headers written into Napatech Socket's transmit buffer.
f. Ip and protocol checksums are calculated if it is indicated with command line arguments.
g. To provide transmit rate user wanted, timestamps of packages calculated. If no transmit rate specified with command line options, packages are set to send at line rate.
h. Transmit buffer is released through transmit port.
i. Generation phase and sending phase repeat with new flow members, till user breaks the loop with Ctrl+C.

Figure 2.2 shows the top level design of BmtGen project.

Figure 2.2 Top Level Design



2.6 USED LIBRARIES

Two main libraries are used to develop proposed packet generator. Libpcap and Napatech API.

The Packet Capture library, libpcap is a portable C/C++ library for network traffic capture that provides implementation-independent access to the underlying packet capture facility provided by the linux operating system [28]. It provides a high level interface to packet capture systems that allows a user-level process to capture packets passing through the network interfaces. All packets on the network are accessible through this mechanism. It also supports saving captured packets to a file, and reading packets from corresponding file. This library serves the proposed traffic generation solution as an ability to read packets from pre-captured files [15]. Following functions are used to be able to read network frames from pre-captured file:

a. pcap_open_offline
b. pcap_datalink
c. pcap_loop

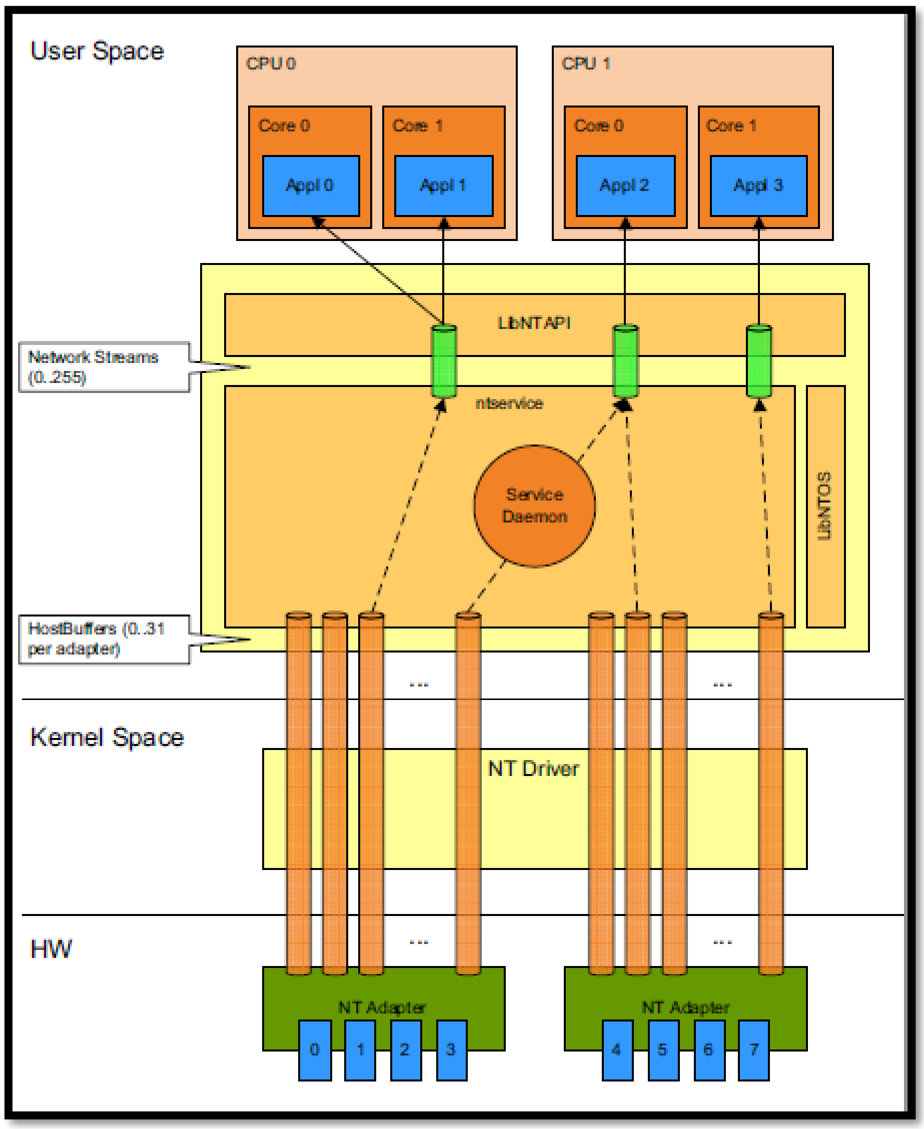
Napatech API, on the other hand, contains third-generation drivers and a software platform for Napatech adapters. It can also scale to new features introduced in the adapter hardware as well as in the driver itself to solve more complicated user scenarios. Major key points of Napatech API is to take advantage of using Napatech cards hardware which provide high performance, low CPU utilization and scalable design [24]. Figure 2.6 shows Napatech Software Suite architecture by showing the various parts of Napatech API.

Napatech cards also support for multiple applications to access the same data. This has been implemented with zero copying. This allows multiple applications to access the same data without having to copy it. The API has only eight function types: Init, NTPL, Open, Close, Get, Release, Read and Write. These eight function types use different parameters depending on which stream type they are used on. To get type checking on all parameters, separate functions have been made for each stream type [8]. This results in 29 functions. We used following functions to be able to send generated traffic data to Napatech cards socket buffer:

a. NT_Init
b. NT_InfoOpen
c. NT_InfoRead
d. NT_InfoClose
e. NT_NetTxOpen
f. NT_NetTxGet
g. NT_NetTxRelease

h. NT_NetTxClose

Figure 2.3 Napatech API architecture



2.7 IMPLIMENTED AND USED FUNCTIONS

Designed and implemented functions are listed below with declaration, starting from first calling function to last one:

a. int main (int argc, char *argv[]) (implemented by us)

When program starts, the main function is called. It is the exact starting point to a program that is executed in an operating system. It has the two-parameter form of parameters. The main function let the user to push a character strings to be passed from the execution environment, via the pointers argv[1] .. argv[argc-1] point at the first characters in each of these strings. argv[0] is the l character of a null-terminated string BmtGen to invoke the program itself. The size of the array pointed to by argv is argc+1, and the last element, argv[argc], is guaranteed to be a null pointer.

b. void read_arguments (int argc, char *argv[]) (implemented by us)
   This function is called with pointer arrays point to each command line argument. It is responsible for parsing command line options and store user defined parameters in configuration structure mentioned 2.2. read_arguments function basically uses getopt function which is POSIX standard.

c. DIR *opendir(const char *name) (implemented in dirent.h conforming to POSIX.1-2001)
   This function opens a directory recursively corresponding to the directory name and path, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

d. struct dirent *readdir(DIR *dirp) (implemented in dirent.h conforming to C99)
   This function returns a pointer to a dirent structure representing the next directory entry in the directory stream pointed to by dirp.

e. char *strcpy(char *dest, const char *src) (implemented in string.h conforming to C99)
   The strcpy function replicates the text indicated by src pointer, including the terminating null byte, to the memory area pointed to by dest. The string dest have to be big enough to store the copy.

f. char *strcat(char *dest, const char *src) (implemented in string.h conforming to C99)
   The strcat function appends string indicated by src pointer to the area pointed to by dest, overwriting the terminating null byte at the end of dest, and also adds a terminating null byte after append operation. The dest string must have enough space for concatenating string.

g. int fputs(const char *s, FILE *stream) (implemented in stdio.h conforming to C99)
   The fputs function writes the string indicated by s pointer to stream, without its terminating null byte

h. size_t strlen(const char *s) (implemented in strlen.h conforming to C99)
   The strlen function returns the length of the string s, excluding the terminating null byte.

i. int fprintf(FILE *stream, const char *format, ...) (implemented in stdio.h conforming to C99)
   The fprintf produce output according to a format and write output to the given output stream;

j. pcap_t *pcap_open_offline(const char *fname, char *errbuf); (implemented in libpcap library)
   The pcap_open_offline function is called to open a "precaptured save file" for reading. The fname pointer specifies the name of the file to open.

k. int pcap_datalink(pcap_t *p) (implemented in libpcap library)
   The pcap_datalink function returns the link layer type for the saved pcap file.

l. int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user) (implemented in libpcap library)

This function processes packets from saved pcap file until given cnt packets are processed, the end of the file reached when pcap_breakloop is called.

m. void save_frames (unsigned char *user, const struct pcap_pkthdr *hdr, const unsigned char *data) (implemented by us)
This function stores every network frame read by pcap_loop and their statistics to pcaps structure mentioned in 2.2.

n. int closedir(DIR *dirp) (implemented in dirent.h conforming to POSIX.1-2001)
This function closes the directory stream associated with dirp. A successful call to this function also closes the file descriptor associated with dirp. The directory stream descriptor is not available after this call.

o. int NT_Init (uint32_t version) (implemented in NTAPI library)
This function must be called prior to any other NTAPI function.

p. int NT_InfoOpen (NtInfoStream_t * hStream, const char *name)
(implemented in NTAPI library)
This function is called to retrieve a handle to an info stream.

q. int NT_InfoClose (NtInfoStream_t hStream) (implemented in NTAPI library)
This function is called to close an info stream.

r. int NT_NetTxOpen (NtNetStreamTx_t *hStream, const char *name, uint64_t portMask, uint32_t NUMA, uint32_t minHostBufferSize) (implemented in NTAPI library)
This function is called to retrieve a TX stream handle.

s. static void *generate_data (conf *traffic_conf) (implemented by us)
This function generates new flows with given configuration options. If IP Address randomization is enabled, ip address is incremented by one for per flow within boundaries given in command line options. If port randomization is

enabled, port number is incremented by one for per flow within boundaries given in command line options. If inner ip randomization is enabled, it is incremented by one for per flow within boundaries given in command line options.

t. int NT_NetTxGet (NtNetStreamTx_t hStream, NtNetBuf_t *netBuf, uint32_t port, size_t packetSize, enum NtNetTxPacketOption_e packetOption, int timeout) (implemented in NTAPI library)
This function is called to acquire a transmit buffer.

u. uint16_t ip_checksum (void* vdata,size_t length) (implemented by us)
This function calculates the IP header checksum for new manipulated header values.

v. unsigned short tcp_checksum (unsigned short *buffer, int size) (implemented by us)
This function calculates the tcp header checksum for new manipulated header values.

w. uint16_t udp_checksum (const void *buff, size_t len, uint32_t src_addr, uint32_t dest_addr) (implemented by us)
This function function calculates the udp header checksum for new manipulated header values.

x. int NT_NetTxRelease (NtNetStreamTx_t hStream, NtNetBuf_t netBuf) (implemented in NTAPI library)
This function releases the netBuf data obtained via NT_TxGet.

y. int NT_NetTxClose (NtNetStreamTx_t hStream) (implemented in NTAPI library)
This function is called to close a TX stream.

2.8 COMMAND LINE ARGUMENTS AND USAGE

In this project, it is planned to take all necessary configuration arguments from the command line, when the program executed. The syntax of running program [4] as follows:

```
./BmtGen [-h][-i <start_ip:end_ip>][-g <gtp_start_ip:gtp_end_ip>]
[-p <start_port:end:port>][-c <on>][-u <on>][-r <bitrate>][-t <transmit_port>]
```

Options:

a. -h option: Prints the help menu to explain user how to use program.

b. -p option: User may enter this option to set port randomization. After this parameter user may enter start and end port numbers to put a boundary for flow generation. Example : -p 0x0:0xFFFF

c. -i option: User may enter this option to set IP address randomization. After this parameter user may enter start and end IP addresses to put a boundary for flow generation. Example : -i 0x0:0xF5A2B3D1

d. -g option: User may enter this option to set gtp-u IP address randomization. After this parameter user may enter start and end IP addresses for gtp protocol to put a boundary for flow generation. Example : -g 0x0A1B2C3D:0xF4E51023

e. -c option: User may set this option if calculation of ip header checksum is requested. The default value of this option off. To re-calculate ip header checksum after every flow generation, user should set this as "-c on"

f. -u option: User may set this option if calculation of protocol checksum is requested. The default value of this option off. To re-calculate protocol header checksum after every flow generation, user should set this as "-u on"

g. -r option: User may enter this option to set bitrate for sending new flow frames. It determines how many megabits per seconds should pass from indicated transmit port. The default value of this option is “line rate”. To limit bitrate, this rate option should set as “-r 100”

h. -t option: User may enter this option to set transmit port number. The default value of this option is “0”. Example: -t 1234

Usage Examples :

a. ./BmtGen -g 0x0A1B2C3D:0xF4E51023

   With this command, BmtGen only changes inner gtp ip addresses between given range.

b. ./BmtGen -i 0x0A1B2C3D:0xF4E51023

   With this command, BmtGen only changes outer ip addresses between given range.

c. ./BmtGen -i 0x0A1B2C3D:0xF4E51023 -p 0x0:0xFFFF

   With this command, BmtGen changes outer ip addresses and ports between given range.

d. ./BmtGen -i 0x0A1B2C3D:0xF4E51023 -g 0x0A1B2C3D:0xF4E51023 -p 0x0:0xFFFF

   With this command, BmtGen changes outer ip addresses, ports and inner gtp ip addresses between given range.

e. ./BmtGen -i 0x0A1B2C3D:0xF4E51023 -g 0x0A1B2C3D:0xF4E51023 -p 0x0:0xFFFF -c on -u on

   With this command, BmtGen changes outer ip addresses, inner gtp ip addresses, ports between given range and calculates checksums for Layer3 and Layer4 headers.

f. ./BmtGen -i 0x0A1B2C3D:0xF4E51023 -g 0x0A1B2C3D:0xF4E51023 -p 0x0:0xFFFF -c on -u on -r 10000

   With this command, BmtGen changes outer ip addresses, inner gtp ip addresses, ports between given range, calculates checksums for Layer3 and Layer4 headers and sends generated data with bitrate 10000Mbit per second.

g. ./BmtGen -i 0x0A1B2C3D:0xF4E51023 -g 0x0A1B2C3D:0xF4E51023 -p 0x0:0xFFFF -c on -u on -r 10000 -t 1245

   With this command, BmtGen changes outer ip addresses, inner gtp ip addresses, ports between given range, calculates checksums for Layer3 and Layer4 headers, sends generated data with bitrate 10000Mbit per second from transmit port 1234.

3. ENVIRONMENT AND TESTS

In this section, test environment is described; test and results are explained and presented. Tests are presented in four main categories which are verification test, load test, feature performance tests and multiprocessor test.

3.1 ENVIRONMENT

The test computer has “Ubuntu 14.04.3 LTS trusty” operating system on it. Ubuntu is a Linux operating system which is based on Debian distribution used mostly for personal computers. Default user interface of Ubuntu is Unity. It is based on free software. The test computer has 3.16.0-24-generic linux kernel on it.

CPU related environmental information as follows:

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
CPU(s):	24
Thread(s) per core:	2
Core(s) per socket:	6
Socket(s):	2
NUMA node(s):	2
Vendor ID:	GenuineIntel
CPU MHz:	2664.091
L1d cache:	32K
L1i cache:	32K
L2 cache:	256K
L3 cache:	12288K

The test computer has one Napatech A/S NT40E3-4-PTP Network card which has 4 ports, each of them 10-Gbps; and the two Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Adapter.

3.2 VERIFICATION TEST

Verification test is a basic approach to examine self-tests to be precisely certain about requirement, specification and functional coverage [20]. The Verification will be granted with two ways. The first one is comparing input packet capture with output packet capture. While BmtGen generates and send packages to output port; another capture tool of Napatech Software Suite listens and captures incoming stream. The captured file should be converted to be compatible to be read by Wireshark program [6]. Wireshark is very famous network protocol analyzer. Wireshark program let the user to see detailed information about what's happening on the network in depth. To convert captured file, capfileconvert tool of Napatech Software Suite is used. The expectation is seeing many different network flows with similar input payload at captured packets.

The second way of verification of functionality of BmtGen is comparing output stats of BmtGen with profiling tool of Napatech Software Suite. BmtGen outputs package counts, byte counts and elapsed time information. All these outputs are compared and verification granted.

To make mentioned tests, following commands are run from three different console applications.

```
sudo ./BmtGen -i 0x0A1B2C3D:0xF4E51023 -t 0
```

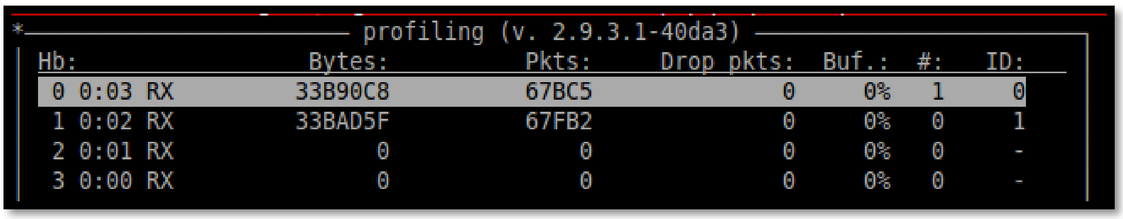
```
./profiling
./capture –s 0 –f file.3gd
./ capfileconvert -i file.3gd -o dest.cap --outputformat=pcap
```

3.2.1 Verification Test Results

After running tests, output capture investigated to check that BmtGen meet the design specification and perform analysis of the modeling input capture payload. It is observed that many different network flows that contains same payload of input capture, ensured functional behavior as expected.

Also comparison between BmtGen statistics and profiling statistics verified the functional accuracy of BmtGen and determined its compliance.

Figure 3.1 Output of profiling tool



```
*─────────────────────────── profiling (v. 2.9.3.1-40da3) ──────────────
 Hb:                   Bytes:        Pkts:     Drop pkts:  Buf.:  #:   ID:
  0 0:03 RX           33B90C8        67BC5             0     0%   1     0
  1 0:02 RX           33BAD5F        67FB2             0     0%   0     1
  2 0:01 RX                 0            0             0     0%   0     -
  3 0:00 RX                 0            0             0     0%   0     -
```

BmtGen Output:

```
=====   STATS   =====
Packet Count  :   425906
Total Bytes   :   54MB 242KB 655 Bytes
```

3.3 LOAD TEST

Load testing is run to determine a system's network behavior under anticipated peak network load conditions. It helps to clarify the maximum operating capacity of BmtGen on network interface ports. When the loads placed on the indicated transmit port is raised beyond normal usage patterns to understand the system's response in aspect of CPU cycle and bitrate correlation.

Load test of BmtGen comprises of sending generated data to particular transmit port with line rate and observing port buffer load.

While BmtGen is manipulating only ip addresses and sending generated data with line rate; profiling tool of Napatech Software Suit will be observed. As a result, buffer load ratio is noted.

Running BmtGen with line rate option: ./BmtGen -i 0x0A1B2C3D:0xF4E51023
This command means send packages on line rate by manipulating IP addresses between 0x0A1B2C3D and 0xF4E51023.

With only one process, BmtGen used 48 percent of CPU, and send generated data with the rate of 8.1 Gigabit per second. On profiling tool, buffer load of corresponding transmit port is observed up to 70 percent.

On behalf of aforesaid results, a suspicion rise. Results have been gave the idea that if the BmtGen traffic generator runs with this rate for longer duration, packet loss could be occur due to socket buffer load. To examine this, BmtGen is run with same input for longer period. After 3 minutes even though packet send rate went up to 8.3 Gigabits per second, buffer load reach to 100 percent and consequently package drops are observed.

In this context it is decided to limit transmit rate and find optimum rate for sending data without package loss. To limit bitrate –r argument is used. After plenty of tests run, 5.4 Gigabits per second rate is determined as a stable transmit ratio.

3.4 FEATURES PERFORMANCE TESTS

Features Performance Tests are run for how each single manipulation feature affects the performance of BmtGen program in terms of transmit rate and CPU usage. These tests are destined to run only one process and one port at the same time. Compared manipulation features are; ip manipulation, port manipulation, gtp ip manipulation, ip header checksum manipulation, and protocol header checksum manipulation.

Manipulated features and results are presented below.

3.4.1 Only Ip Address

```
./BmtGen -i 0x0A1B2C3D:0xF4E51023 -r 5400 -t 0
```

Transmit Rate: 5.4 Gigabits per second
CPU Usage: 45 percent

IP addresses are changed between 0x0A1B2C3D and 0xF4E51023 and manipulated network data sent with 5.4 Gigabits per second bitrate on port 0. As a result, given transmit rate is granted and 45 percent of CPU usage is observed while operation runs.

3.4.2 Ip Address And Port

```
./BmtGen -i 0x0A1B2C3D:0xF4E51023 -p 0x0:0xFFFF -r 5400 -t 0
```

Transmit Rate: 5.4 Gigabits per second
CPU Usage: 55 percent

IP addresses are changed between 0x0A1B2C3D and 0xF4E51023 and port numbers are changed between 0x0 and 0xFFFF. Manipulated network data sent with 5.4 Gigabits per second bitrate on port 0. As a result, given transmit rate is granted and 55 percent of CPU usage is observed while operation runs.

3.4.3 Only Gtp Address

```
./BmtGen -g 0x0A1B2C3D:0xF4E51023 -r 5400 -t 0
```

Transmit Rate: 5.4 Gigabits per second
CPU Usage: 50 percent

GTP IP addresses are changed between 0x0A1B2C3D and 0xF4E51023 and manipulated network data sent with 5.4 Gigabits per second bitrate on port 0. As a result, given transmit rate is granted and 50 percent of CPU usage is observed while operation runs.

3.4.4 Gtp Address And Port

```
./BmtGen -g 0x0A1B2C3D:0xF4E51023 -p 0x0:0xFFFF -r 5400 -t 0
Transmit Rate: 5.2 Gigabits per second
CPU Usage: 61 percent
```

GTP IP addresses are changed between 0x0A1B2C3D and 0xF4E51023 and port numbers are changed between 0x0 and 0xFFFF. Manipulated network data sent with 5.4 Gigabits per second bitrate on port 0. As a result, given transmit rate could not be granted. 5.2 Gigabits per second bitrate and 55 percent of CPU usage is observed while operation runs.

3.4.5 Ip Address And Gtp Address

```
./BmtGen -i 0x0A1B2C3D:0xF4E51023 -g 0x1B0A3D2C:0xFFEE4573 -r 5400 -t 0
Transmit Rate: 4.9 Gigabits per second
CPU Usage: 65 percent
```

IP addresses are changed between 0x0A1B2C3D and 0xF4E51023 and GTP IP addresses are changed between 0x1B0A3D2C and 0xFFEE4573. Manipulated network data sent with 5.4 Gigabits per second bitrate on port 0. As a result, given transmit rate could not be granted. 4.9 Gigabits per second bitrate and 65 percent of CPU usage is observed while operation runs.

3.4.6 Ip Address, Gtp Address And Port

```
./BmtGen -i 0x0A1B2C3D:0xF4E51023 -g 0x1B0A3D2C:0xFFEE4573 -p 0x0:0xFFFF
-r 5400 -t 0
Transmit Rate: 4.8 Gigabits per second
```

```
CPU Usage: 71 percent
```

IP addresses are changed between 0x0A1B2C3D and 0xF4E51023 and GTP IP addresses are changed between 0x1B0A3D2C and 0xFFEE4573 and port numbers are changed between 0x0 and 0xFFFF. Manipulated network data sent with 5.4 Gigabits per second bitrate on port 0. As a result, given transmit rate could not be granted. 4.8 Gigabits per second bitrate and 71 percent of CPU usage is observed while operation runs.

1.4.7 Ip Address And Ip Checksum

```
./BmtGen -i 0x0A1B2C3D:0xF4E51023 -r 5400 -t 0 -c on
Transmit Rate: 5.2 Gigabits per second
CPU Usage: 59 percent
```

IP addresses are changed between 0x0A1B2C3D and 0xF4E51023 and IP header checksum is asked to be changed by BmtGen. Manipulated network data sent with 5.4 Gigabits per second bitrate on port 0. As a result, given transmit rate could not be granted. 5.2 Gigabits per second bitrate and 59 percent of CPU usage is observed while operation runs.

1.4.8 Ip Address, Port, Ip And Port Checksum

```
./BmtGen -i 0x0A1B2C3D:0xF4E51023 -p 0x0:0xFFFF -r 5400 -t 0 -c on –u on
Transmit Rate: 4.9 Gigabits per second
CPU Usage: 81 percent
```

IP addresses are changed between 0x0A1B2C3D and 0xF4E51023 and port numbers are changed between 0x0 and 0xFFFF. IP header checksum and protocol header checksum are asked to be changed by BmtGen. Manipulated network data sent with 5.4 Gigabits per second bitrate on port 0. As a result, given transmit rate could not be granted. 4.9 Gigabits per second bitrate and 81 percent of CPU usage is observed while operation runs.

1.4.9 Ip Address, Gtp Address, Port, Ip And Port Checksum

```
./BmtGen -i 0x0A1B2C3D:0xF4E51023 -g 0x1B0A3D2C:0xFFEE4573 -p 0x0:0xFFFF
-r 5400 -t 0 -c on –u on
Transmit Rate: 4.2 Gigabits per second
CPU Usage: 93 percent
```

IP addresses are changed between 0x0A1B2C3D and 0xF4E51023 and GTP IP addresses are changed between 0x1B0A3D2C and 0xFFEE4573 and port numbers are changed between 0x0 and 0xFFFF. IP header checksum and protocol header checksum are asked to be changed by BmtGen. Manipulated network data sent with 5.4 Gigabits per second bitrate on port 0. As a result, given transmit rate could not be granted. 4.2 Gigabits per second bitrate and 93 percent of CPU usage is observed while operation runs.

1.4.10 Features Performance Test Results

After careful examination of features tests, it is observed that in aspect of CPU usage, GTP IP address manipulation is the most expensive operation, which is followed by respectively; IP address manipulation, IP header checksum operation, protocol checksum operation and finally port number manipulation. All feature test results are represented briefly in Table 3.1.

Table 3.1 Feature Test Results

Feauture Test	Performance Values	
	Bitrate (Gbit/ps)	CPU Usage (%)
IP	5,40	45
IP + Port	5,40	55
GTP	5,40	50
GTP + Port	5,20	61
IP + GTP	4,90	65
IP + GTP + Port	4,80	71
IP + Ipchksum	5,20	59
IP + Port + Ipchcksum+ Portchcksum	4,90	81
IP + GTP + Port + Ipchcksum+ Portchcksum	4,20	93

3.5 MULTI PROCESS TEST

Multiprocessing is the use of more than one CPUs within a one computer system [22]. This term also express the ability of a program to support more than one processor and/or the ability to allocate tasks between them. Even-though BmtGen has no inter process communication mechanism, it may be used as multiple process pinned to multi-processor with the help of command line arguments. CPU pinning make it possible to binding and unbinding of a process, in our example BmtGen process, to a CPU, so that the process execute only indicated particular CPU. This can be thought as a modification of the native central queue scheduling in Linux operating system. Each item in the queue has a tag indicating its pinned processor. A practical example of processor affinity is executing multiple instances of a non-threaded application, such as BmtGen.

To take advantage of processor affinity using BmtGen, it is important to change port argument of each BmtGen process. Also manipulation range should be decided carefully to prevent flow overlap. For CPU affinity, taskset program is used.

3.5.1 Multi-process Test Results

Testing with two process by binding processes to different CPUs made with following arguments

```
./BmtGen -i 0x00000000:0x85858585 -r 5400 -t 0
./BmtGen -i 0x85858586:0xFFFFFFFF -r 5400 -t 1
```

5.4 Gigabits per second transmit rate is observed in both transmit port; total 10.4 Gigabits per second transmit rate is met. When BmtGen is tested with four processes pinned to different CPUs, total transmit rate reached 22.2 Gigabits per second. It is predictable that BmtGen can easily reach more than 40 Gigabits per seconds if eight processes used, yet since Napatech card on system has only 4 ports, it was not applicable to test BmtGen with more than 4 processors.

4. DISCUSSION

As it is already mentioned in Introduction section, increasing personal use of new network devices force operators to use DPI applications for better understanding traffic profile. In this respect, operators developed deep packet inspection applications for optimizing their networks performance. The proposal in this thesis offers a solution to test DPI application developed for LTE networks. To be able to test those applications, traffic generators dedicated to particular protocols and particular conditions discussed above, are needed.

It is investigated that previous works on this area and previous traffic generator products could not address conditions. Hardware based traffic generators are unable to intersect LTE nodes and also create extra costs to operators. Software traffic generators which create dummy or synthetic traffic are mostly served for stress testing and also not proper for testing payload based DPI operations. Traffic generators which algorithm depends on replay, aim to inject previously collected packet captures to the network. Although this algorithm gives the flexibility to serve near-real time traffic which could be best fit for corresponding DPI applications, packet captures should be huge sized to be able to create many different flow and test correlation functions of DPI applications. When it comes to work with huge sized network captures, these traffic generators cause segmentation faults on system.

To overcome all this problems and answer all demands proposed solution BmtGen is developed and implemented. Verification tests show that BmtGen solution is precisely certain about requirements, specification and functional coverage. Load test clarified the maximum operating capacity of BmtGen on network interface cards. Features Performance Tests pointed how each single feature affects the performance of BmtGen in terms of transmit rate and CPU usage. Finally with Multi-process tests, it is assured that BmtGen can reach requested transmission rates for testing DPI applications. After all tests BmtGen has proved itself to be a proper traffic generating and testing tool for DPI application developed for LTE networks.

5. CONCLUSION

In this thesis, a new solution for generating different flows with real time payload is proposed to test dpi tools which implemented for LTE networks. Requirements have been analyzed. The BmtGen application has been planned, designed, implemented and finally tested, to figure out whether the requirements are met or not. The source code of BmtGen is published at open source community github.

After completing all test phases that explained in detail at above section, following results are observed. BmtGen application had not been encountered any stack overflow error or segmentation fault as other software based traffic generators did. The main distinctive property of BmtGen is observed when output of BmtGen is sniffed. Wide range of flows, with real packet payload, is detected. This outcome satisfies one of the main goals.

While fulfilling feature requirements, BmtGen did not fail in the manner of performance. Both network socket performance and CPU usage performance was observed at optimum level, thus it can be seen clearly after tests, BmtGen can be easily used to test DPI tools run on LTE networks, without any major drawback.

BIBLIOGRAPHY

[1] 3GPP TS 36.101 (October 2011) User Equipment (UE) Radio Transmission and Reception, Release 10

[2] A.Callado, C.Kamienski, G.Szabo, B.P.Gero, J.Kelner, S.Fernandes, D.Sadok, A survey on internet traffic identification, Commun Surv Tutor, IEEE11 (2009) 37–52

[3] Available online: http://www.ixiacom.com/, last accessed at May 10, 2016

[4] BmtGen, available online: https://github.com/kursatkobya/trafficgen, last accessed at May 11, 2016.

[5] C. Schroder, Measure Network Performance With ipef, Enterprise Networking Planet, 2007.

[6] Combs, Gerald. "Wireshark-network protocol analyzer." Version 0.99 5 (2008)

[7] D. E. Comer, Internetworking with TCP_IP, 2013

[8] Deri, Luca. "nCap: Wire-speed packet capture and transmission." End-to-End Monitoring Techniques and Services, 2005. Workshop on. IEEE, 2005.

[9] G. Salmon, M. Ghobadi, Y. Ganjali, M. Labrecque, J.G. Steffan, NetFPGA-based precise traffic generation, in: Proc. NetFPGA Developers Workshop' 2009.

[10] "GCC Releases". GNU Project. Retrieved December 27, 2006

[11] HSU, Chung-Hsing; KREMER, Ulrich. IPERF: A framework for automatic construction of performance prediction models. In: Workshop on Profile and Feedback-Directed Compilation (PFDC), Paris, France. 1998.

[12] Iperf, available online: http://sourceforge.net/projects/iperf/, last accessed at May 10, 2016.

[13] J. Jaeger, FPGA-based prototyping grows up, Electronic Engineering Times (518) (2008).

[14] J.Sommers, P.Barford, Self-configuring network traffic generation, Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, IMC'04, 2004, pp.68–81.

[15] Jacobson, V., C. Leres, and S. McCanne. "libpcap, Lawrence Berkeley Laboratory, Berkeley, CA." Initial public release June (1994).

[16] K.Vishwanath, A.Vahdat, Swing: realistic and responsive network traffic generation, IEEE/ACM Trans Netw 17 (2009) 712–725.

[17] List of generators online: http://traffic.comics.unina.it/software/ITG/link.php, last accessed at May 10, 2016.

[18] M. C. Weigle, P. Adurthi, F. Hernández Campos, K. J. Kevin, F. D. Smith, Tmix : A tool For Generating Realistic TCP Application Workloads In ns-2, SIGCOMM Comput.Commun, (2006) 65–76.

[19] M. Fourment, M. R. Gillings, A comparison of common programming languages used in bioinformatics, BMC Bioinformatics, (2008).

[20] McCluskey, Edward J. "Verification Testing&# 8212; A Pseudoexhaustive Test Technique." Computers, IEEE Transactions on 100.6 (1984): 541-546.

[21] MGEN, available online: http://cs.itd.nrl.navy.mil/work/mgen/, last accessed at May 10, 2016.

[22] Mike Ebbers; John Kettner; Wayne O'Brien; Bill Ogden (2012). Introduction to the New Mainframe: z/OS Basics. IBM.

[23] N.Secchi, R.Bonelli, S.Giordano, G.Procissi, BRUTE: a high performance and extensible traffic generator, Proceedings of the SPECTS'05, Philadelphia, PA, USA, 2005, pp.839–845.

[24] NT40, Napatech. "NT20." (2012).

[25] Scopatz, Anthony; Huff, Kathryn D. (2015). Effective Computation in Physics. O'Reilly Media, Inc. p. 351. ISBN 9781491901595. Retrieved 20 April 2016.

[26] Stefania Sesia, Issam Toufik, Matthew Baker, LTE - The UMTS Long Term Evolution: From Theory to Practice, Second Edition, 2011

[27] Stevens, UNP page. 707

[28] Tcpreplay, available online: http://tcpreplay.synfin.net/, last accessed at May 10, 2016.

[29] TG, available online: http://www.postel.org/tg/tg.html, last accessed at May 10, 2016.

[30] TURNER, Aaron; BING, Matt. Tcpreplay. 2005.

[31] Olsson & Mulligan, EPC and 4G Packet Networks, 2nd Edition, 2012.

[32] W.Feng, A.Goel, A.Bezzaz, W.Feng, J.Walpole, TCPivo : a high performance packet replay engine, Proceedings of the ACM SIGCOMM Workshop on Models, methods and tools for reproducible network research, Karlsruhe, Germany, 2003, pp.57–64.