UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

MASTER THESIS

HIGH PRECISION PACKET TIME-STAMPING USING NETFPGA 10 G PLATFORM

Antonios Gavaletakis
(antonios.gavaletakis@estudiante.uam.es)

SEPTEMBER 2014

HIGH PRECISION PACKET TIME-STAMPING USING NETFPGA 10G PLATFORM

AUTOR: Gavaletakis Antonis
TUTOR: Dr. Gustavo Sutter

High Performance Computing and Networking
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
September 2014

Abstract

Abstract — High precision network measurements is an area with high interest as the performance of the networks affects the quality and the cost of a service between a Network Service Provider (NSP) and the costumer. The increase of the network speed leads the measurements of the software system to be unreliable even though their low cost and the high configurability. The solution for high network performance measurement at high network speed is hardware system that can guarantee standard high performance.

The NetFPGA is an open source low-cost platform based on networks that permits to implement network system easily due to the wide reference components that offers. The second version of the NetFPGA platform designed by the Stanford University has four 10GigE SFP+ interfaces and a powerful FPGA providing the ability to implement network system over copper and optic fiber at 1Gbps and 10Gbps.

This NetFPGA 10G project can measure the network parameters at high precision with the technique of the time stamping. A GPS system guarantees the high precision of the time. The dynamically generation of back-to-back packets gives the flexibility for measurements without any recaptured flows that no other system provides. The save of the captured packets gives the possibility of off-line further analysis of the network.

Index Terms — NetFPGA, network measurement, high precision, packet time-stamping, GPS

Resumen

Resumen — Medidas de red de alta precisión es un área de gran interés de como el desempeño de las redes afecta la calidad y el costo del servicio entre un proveedor de servicios de red (NSP) y el consumidor. El incremento de la velocidad de las redes lleva a que las mediciones por software sean poco fiables a pesar de su bajo coste y alta configurabilidad. La solución para mediciones de alto rendimiento en redes de alta velocidad son sistemas hardware que pueden garantizar alta rendimiento estándar.

La NetFPGA es una plataforma de código abierto de bajo coste basado en redes que permite implementar sistemas de red con facilidad debido al gran soporte de componentes de referencia que ofrece. La segunda versión de la plataforma de NetFPGA desarrollada por la Universidad de Stanford tiene cuatro interfaces de 10GigE de tecnología SFP+ y una potente FPGA que permite implementar sistemas de red con conexiones de cobre y de fibra óptica de 1Gbps y 10Gbps.

Este proyecto de NetFPGA 10G puede medir parámetros de red con alta precisión con la técnica de marca de tiempo (time-stamping). Un sistema GPS garantiza alta precisión de tiempo. La generación dinámica de paquetes consecutivos da la flexibilidad para mediciones sin reproducir tráfico anteriormente capturado, cosa que otros sistemas no pueden hacer. El guardado de paquetes generados da la posibilidad de futuros análisis sin repetir los experimentos (análisis off-line).

Palabras Clave — NetFPGA, network measurement, high precision, packet time-stamping, GPS

Agradecimientos

En primer lugar me gustaría agradecer a mi tutor Gustavo Sutter su inestimable ayuda durante la realización de este proyecto. Él fue quien me aconsejo durante todo el trabajo y quien expuso la principal idea para implementar el sistema que he desarrollado y me apoyo durante estos meses de investigación.

Mención especial a los profesores Sergio López Buedo e Jorge López de Vergara por sus sabios consejos e importantes recomendaciones.

En estos agradecimientos no podrá faltar la maravillosa ayuda que me han ofrecido durante mi estancia en este laboratorio mis compañeros. Sin ellos el día a día hubiese sido, sin duda, más aburrido y aun me encontraría buscando en google alguna solución. En especial a Rafa Leira por ofrecer me esta plantilla de Latex, y por los innumerables veces que me hizo más ameno el camino en el tren de cercanías. A Mario Ruiz le agradezco su ayuda y amistad dentro y fuera del laboratorio y a José Zazo Rollón y David Muelas su colaboración desde el primer momento en la implantación de este proyecto. Y para mi familia que desde Grecia me ha apoyado les dedico las siguientes palabras.

Ευχαριστώ τους γονείς μου και τον αδερφό μου Μάριο για την συνεχή υποστήριξη τους όλο αυτό το διάστημα που βρισκόμουν στην Ισπανία.

Τοις τολμώσιν η τύχη ξύμφορος.
(Η τύχη βοηθάει τους τολμηρούς.)
Θουκυδίδης 460-394 π.Χ

Contents

List of Tables

List of Figures

1 Introduction

The network's performance always is getting higher and higher. The last 20 years the wired networks speed is 1.000 times faster, starting with 10BASE-T at 10 Mbit/s speed at the beginning of 90s [1] and nowadays achieving 10Gbps speeds. The Ethernet evolution includes higher bandwidth, improved media access control methods, and different physical media. This high speed evolution leads the packet processing like packet classification, manipulation, and forwarding to be done at very high rates [2].

Network performance is the main criterion for the service quality and service cost between the Network Service Provider (NSP) and the costumer. The Service Level Agreement (SLA) that specifies the quality level of service that the service NSP and the client are agreed should be respected and monitoring by the NSP. SLA monitoring involves the monitoring the performance status of the offered service and provide relevant information to the service level management system. The main network performance metrics (NPM) that should be measured are Availability (Connectivity, Functionality), Loss (One Way Loss, Round Trip loss), Delay (One-Way delay, RTT delay) and Utilization (Capacity, Bandwidth and Throughput). Consequently, the careful networks' measuring with high precision is crucial [3].

The NPM are measured by various network monitoring technologies either software or hardware. Depending on goals of the application, each one of these solutions has advantages and disadvantages. For network measurements at high precision and speed the software solutions have a lot of disadvantages. First of all, the software systems do not provide high precision as they are depended on the hardware performance of the host computer that are executed. What is more, for high precision network measurements a high performance computer is demanded which results in high price [4]. In addition, the high rate measurement demands a lot of computer resources with high CPU load witch results to high power consumption that is an important issue nowadays. On the other side, the hardware solution guarantees high performance as the execution of the system is in depended from the computer resources. The majority of the times the hardware systems are executed without the use of any computer. The high measurement rates do not need more computational resources so the power consumption is always unchanged and low.

The use of a FPGA for network measurement at high rates with high precision is the best solution. A field-programmable gate array (FPGA) is a reprogrammable hardware that guarantees high performance. With unprecedented logic density increases and a host of other features, such as embedded processors, DSP blocks, clocking, and high-speed serial at ever lower price points, FPGAs are a compelling proposition for system of high speed network measurement. In addition, FPGA systems are lower cost solution comparing with hardware commercial system or software solution with high performance computer providing the same or better results. The FPGA power consumption is always the same, at low level comparing to computer power consumption as the execution code on the FPGA always consumes the same energy [5].

This thesis has been inspired by the need for an affordable high precision network measurement at high rate program using packet time-stamping. This can be achieved by implementing a configurable traffic generator which time-stamps the packets with high precision using a GPS system. In addition, as high precision measurements are needed, both the part of the packets generation and the part of the receiving and time-stamp analyzing packets should be implemented in advance. The implementation is done in the NetFPGA 10G (network Field-Programmable Gate Array) platform that is the low-cost reconfigurable hardware platform optimized for high-speed networking. It includes all the logic resources, memory, and Gigabit Ethernet interfaces necessary to build a packet generator and a high precision packet receiver [6].

To date, there is a wide variety of tools generating traffic - both commercial and open-source solutions, either in software or in hardware [7–9] . Unfortunately, they have important disadvantages (inaccurate measurements, high cost).

To conclude, all these disadvantages of inaccurate measurements and high power consumption that the software implementations have, as well as the high cost of the commercial products can be solved with this implementation of gigabit high precision time-stamping network measurement project on the NetFPGA 10G platform. The goal of this project is to implement a packet generator that provides reliable time-stamp flow generation without the pre knowledge of other flows with low power because of the technology of the FPGA and a receiver that can receive, time-stamp and store tha packets to the host's hard drive.

1.1 Objective

This work is focused in the network performance measurement at high rates with high precision. The aim is to build a system that will measure the performance parameters of throughput, packet losses, jitter and one-way delay using the technique of the packet time-stamping. For high precision of the measurements the new generation of the NetFPGA 10G of the Stanford is used. The implemented system in the platform of the NetFPGA 10G is planned to be separated into two parts: the packet generation part and the packet reception and analysis.

The goal of the sender is to generate back-to-back marked UDP packets with generation time and sequenece number of parameterized MAC, IP, UDP headers and payload size. For high time-stamping precision a GPS system will be used that guarantees the precise announcement of the seconds. The generation of the packets will be done dynamically on the card without the replication of pre-saved information. The part of the receiver is planned to be responsible for the reception, the time-stamping and the store of the received packets at the hard disc of the computer for further processing. The aim for the time-stamping of the received packets is to be implemented using a GPS system. The goal is the generating part as well as the receiving part to be implemented in the same NetFPGA card in order to avoid problems of clock synchronization.

Up to date, there have been several commercial implementations in hardware like Network latency measurements of APCON [10] as well as in university and researching communities like HATS: High Accurate Time stamping System Based on NetFPGA [11] and OSNT:Open Source Network Tester [12]. Each one of these implementations has disadvantages that our system solves. As the aim is to implement a lower cost system as commercial product, the academic hardware of the NetFPGA is used as it has low cost and a powerful FPGA for implement big and complex systems. The implementation of [12, 13] needs a large database of stored net-flows as they replicated the stored flows which are as important limitation. Aim of this implementation is not to have this limitation, by generating and the sending of the flows without need of a prefabricated filing base. The goal is the generated packets to be produced dynamically on the card only from the preferences of the user without the pre-capture of any traffic flow.

1.2 Motivation

Nowadays there is need for high speed network measurements with high precision. There are several implementations for measuring the network performance with the technique of the packet generation in software [9] and in hardware [7, 8] as well. Network measurment systems with the technique of the time-stamping are not so spread. The demand of a hardware implementation in time-stamping technique is really high as high precision in time-stamping is needed. Hardware implementation always yields high performance. The speed-up of the hardware implementations compared to the software implementations is substantial as the low-level programming better exploits the hardware. The already existed hardware implementations of the time-stamping technique or are limited to measure network of 1Gbps [14] or need a big bang of pre-captured flows for replicating in order to measure the network.

As it is observed there is not a low-cost system that could generate dynamically packets without the pre-captured and stored flows and mark the packets in order to store the time-stamped testing packets and measure networks at 10Gbps either with optical fiber connection or copper connection.

A possible solution could be an implementation in NetFPGA 10G for the packet generation, that combines high performance, low consumption of energy and low cost and gives the possibility to be implemented systems that can process packets at 10Gbps either on optical fiber or copper connection. For the time-stamping the best solution is GPS system that guarantees high precision of timing. The high NetFPGA performance is achieved as the functionality of the system is programmed in the powerful FPGA of the card and does not need the use of a power computer like the software implementations. The card of the NetFPGA 10G has four SFP+ interface that give the possibility to connect either optical fiber cables or copper cables that works at 10Gbps.

1.3 Thesis Organization

The rest of the thesis is organized as follows; Chapter 2 presents the NetFPGA 10G platform, how a new project is designed and the OSTN project. The measuring network parameters are presented at chapter 3. The design and the architecture of the hardware and software implementation of our system are discussed in Chapter 4. The results of the implementation are presented in Chapter 5. Chapter 6 summarizes and concludes this thesis by listing future enhancements that can be implemented.

2 NetFPGA 10G and OSTN

2.1 The NetFPGA board

The NetFPGA platform is used for the implementation of this system. NetFPGA (network Field-Programmable Gate Array) is the low-cost reconfigurable hardware platform optimized for high-speed networking that the Stanford University is developed. It is an open source hardware and software platform designed for research and teaching. Currently there are two platforms: NetFPGA-1G (1G) and the NetFPGA-10G (10G) [6].

For this implementation, the second newer version of the NetFPGA is used. The card has four 10GigE SFP+ interfaces, a PCI Express interface to the host (Gen2 x8 channels), and a Xilinx Virtex-5 TX240T FPGA for the implementation of the functionality logic. The board has SRAM and DRAM (27 MBytes QDRII SRAM, 288 MBytes RLDRAM-II) memories and a high bandwidth expansion connector for daughter-cards. As the card supports SFP+ extensions, the system can work with fiber optical cables or copper cables without any changes at rates of 10Gbps and 1Gbps. The x8 PCI Express Gen 2 that provides 5Gbps per lane is suitable for writing the received flows to the host's hard disk through DMA (Dynamical Memory Access). The powerful FPGA Xilinx Virtex-5 XC5VTX240TFFG1759 is big enough to be applied the blocks of the microprocessor, the PCIe endpoint with DMA as well as the building blocks of the system. The big SRAM and the bigger DRAM memory of the card are suitable for or storing forwarding table data and packet buffering respectively. All these components are showed at figure 2.2. In the following chapters it is explained how these components are used for this project.

The community of the NetFPGA apart from the hardware of the card offers a plenty of implemented reference designs like NIC in 10G mode with all 4 ports active, switch at 10Gbps and router at 10Gbps and much more. In advance, a big bank of basic building blocks of all future designed is offered that implement basic functionalities of the card like PCIe endpoint with DMA, a microprocessor, UART communication with the host, Ethernet interface core modules, a register system, etc. The software code is available also. The software part contains the reference NIC 10G driver. Apart from the basics functionalities of the driver that are to read and write the packets are coming from the network interfaces and the host respectively, is to read and write the registers of the card. This is achieved through the PCIe and the DMA that are implemented in hardware on the card and in software on the driver. The writing of the card's registers allows to parameterize the functionality of the card and the reading of the card's registers informs the software and the user about the state of the card.

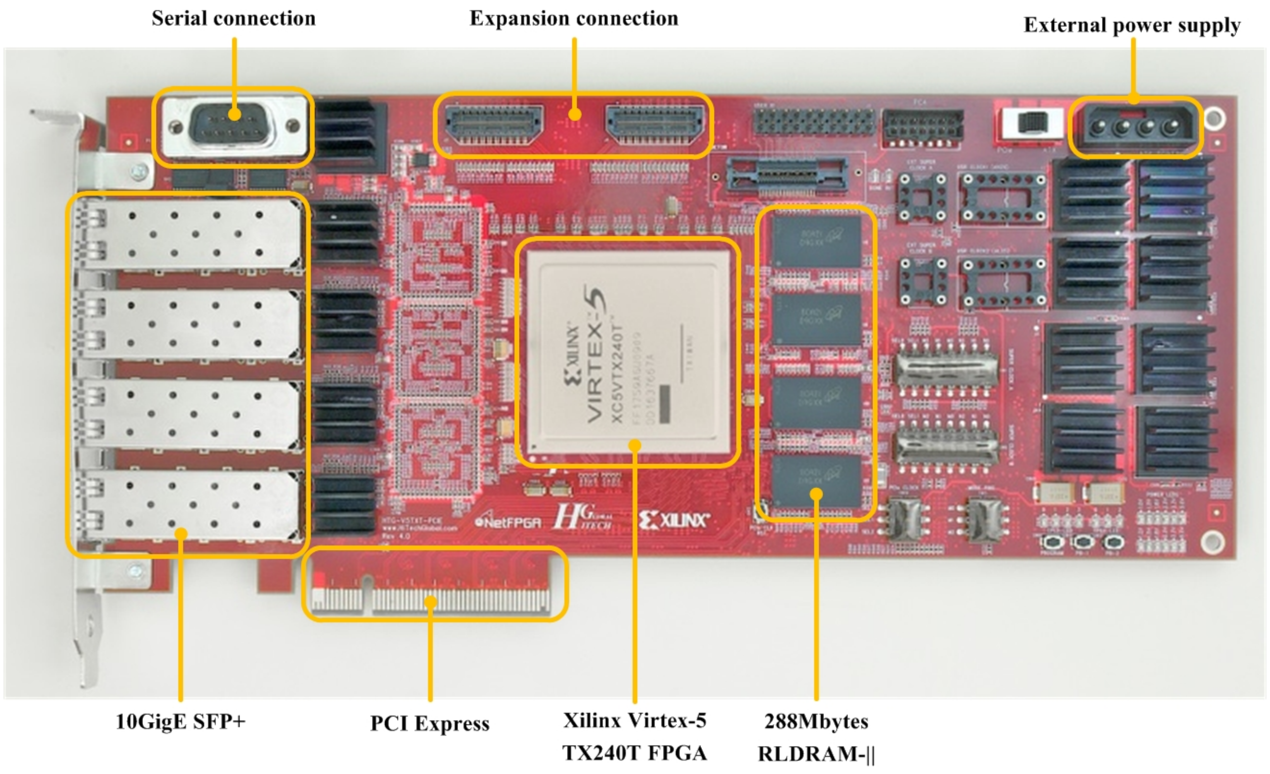


Figure 2.1: NetFPGA 10G card components

2.2 How to design into NetFPGA project

All the code that it is implemented for the NetFPGA projects is open and free. All the available reference projects implement the basic infrastructures of the cards that are the communication between card and host and the reception and analyses of the network packets. Consequently, the design and implementation of new project that is relative to network implementation, with the NetFPGA platform is easier than begin to design and implement with any other FPGA platform.

The reference NIC 10G is the reference project that all the other project are based on it. Understanding these hardware modules is essential in making the most of the available designs. This design is a pipeline where each stage is a separate module as the figure 2.2 illustrates.

The packets that are receives in from the network enter the device through the nf10_10g_interface module, which is an IP that combines Xilinx XAUI and 10G MAC IP cores, in addition to an AXI4-Stream adapter. The received packets are converted from XAUI protocol to AXI4-Stream. In transition case the opposite conversion is executed. Five instances of this module are used, four for each physical network interface and one for the DMA transactions.

Next, the packets enter the input arbiter module. The five flows that are coming from the five nf10_10g_interface instances are stored temporarily into their FIFOs and then forward to the next module by selecting sequentially a flow each time. The round-robin arbiter selects each time a new non-empty queue to forward to the output port lookup module.

The output port lookup module receives the packets and decides which port a packet goes out of. The look up scheme is simple as when the packet comes from the network it forwards it to the CPU and vice versa based on the source port indication.

According to the packet's destination it is selected, the nf10_bram_output_queues module stores it to output queues. There are five output queues as four are for the 10G ports and one for the DMA block

The DMA module serves as a DMA engine. Its purpose is to read and write packets that are receiving or sending from the CPU through the PCIe interface as well the host can access the card registers. The Xilinx Microblaze subsystem that is implemented in the FPGA is used for initializing the cores of the card and the host to communicate through UART with the card for debugging.

The new modules of a new NetFPGA design should be inserted and connected between the input and output queues. At the point of the input/output queues the information of the received/transmit packet can be processed. The basic modules of Microblaze, DMA and AXI interconnection should be used for having the main functionalities of communication with the host. The standards of the AXI4-STREAM and AXI-Lite make simpler the design and the interconnection of the modules.

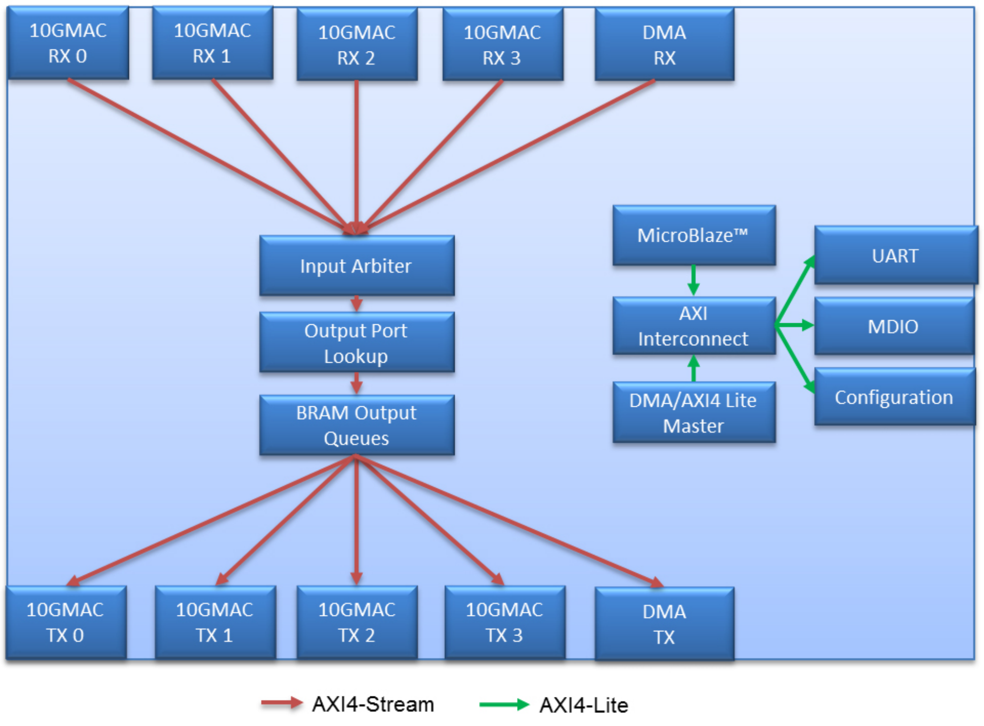


Figure 2.2: Reference pipeline of the Reference NIC project

2.3 The OSNT project

The Open Source Network Tester (OSNT) is an open-source project based on NetFPGA 10G platform that provides a traffic generator and a capture system with 4x10Gbps SFP+ interfaces. It is a combination of the OSNT Traffic Generator and OSNT Traffic Monitor features into single FPGA device and single card. The OSNT's packet generator part can generate packets by reproducing pre-loaded PCAP traces. For each port it can reproduce only one trace in a continuous loop. The dimension allowed of the maximum trace dimension depend strictly depends on the SRAM resources available on board. The packet generator does not support time-stamping of the generated packets. The OSNT traffic monitor provides packet capture at full line-rate. In addition allows packet filtering permitting selection of traffic-of-interest. The captured packets are marked with the arrival time with high precision and accurate. The statistics can gather and store at the hard disk of the computer for further analysis.

2.3.1 OSNT traffic monitor

This system receives the incoming packets and stores them at the hard disk of the computer. It can capture packets at full line-rate of 10Gbps with high precision and accurate. The figure 2.3 illustrates the architecture of the monitoring pipeline. The modules of the Physical interfaces that are before the receive queues, are changed properly for time-stamping the incoming packets, so the incoming packets are time-stamped when exactly enter the pipeline data path of the card. The four incoming flows are aggregated into one from at the Input Arbiter module and sent to the Core Monitoring module.

The Core Monitoring module is responsible for aggregating statistics and sent it to the host and to select and cut the packets that will be sent to the hard disk through the DMA and the PCIe. In details, the Statistics Collector module receives the packets and calculates the average throughput in bits per second (bps) and packets per second (pps) and count the packet number of each protocol (UDP, TCP, Ethernet.). The filtering stage is done by the modules Header Extraction, TCAM and Decision Module. The 5-tuple (protocol, IP address pair and layer four port pair) extraction is performed using an extensible packet parser able to recognize VLAN packets. Only packets that are matched to a rule are sent to the software, while all other packets are dropped. Another mechanism records a fixed-length part of each packet (sometimes called a snap-length) along with a hash of the dropped part. The goal of the dropped part hash is that if a user is interested in all packets on all interfaces it is possible to exhaust the host resources.

The software is divided into two parts; the GUI and the driver with the relevant program for receiving and packets. The python-based GUI allows the user to interact with the HW components (e.g., enable cut/hash, set filtering rules, check statistics). In addition GUI informs the user with the statistics of the card (number of packet, bits per second, packets per second, etc.). For the storing of the receive packets it is needed a kernel level program (driver) and a user level program. The driver receives the packets that the card sends through the PCIe using the DMA mechanism. The driver stores temporary the packets at a share kernel/user memory. The device driver secures performance by bypassing the Linux TCP/IP stack. A C-based application that comes with it records the received traffic that is temporarily stored in the shared memory in either PCAP or PCAPNG format to the disk.

For the accurate time-stamping it is used a GPS external system that it is connected with the NetFPGA card on AU19 General Purpose Input/Output (GPIO) pin. This pin provides a stable pulse-per-second (PPS) signal such as that derived from a GPS receiver permits both high long-term accuracy and the synchronization of multiple OSNT elements. Each packet is appended with a 64-bit timestamp. The upper 32-bits count seconds, while the lower 32-bits provide a fraction of a second with a maximum resolution of approximately 233ps. The practical prototype resolution is 6.25ns because of the 160MHz clock of the FPGA.

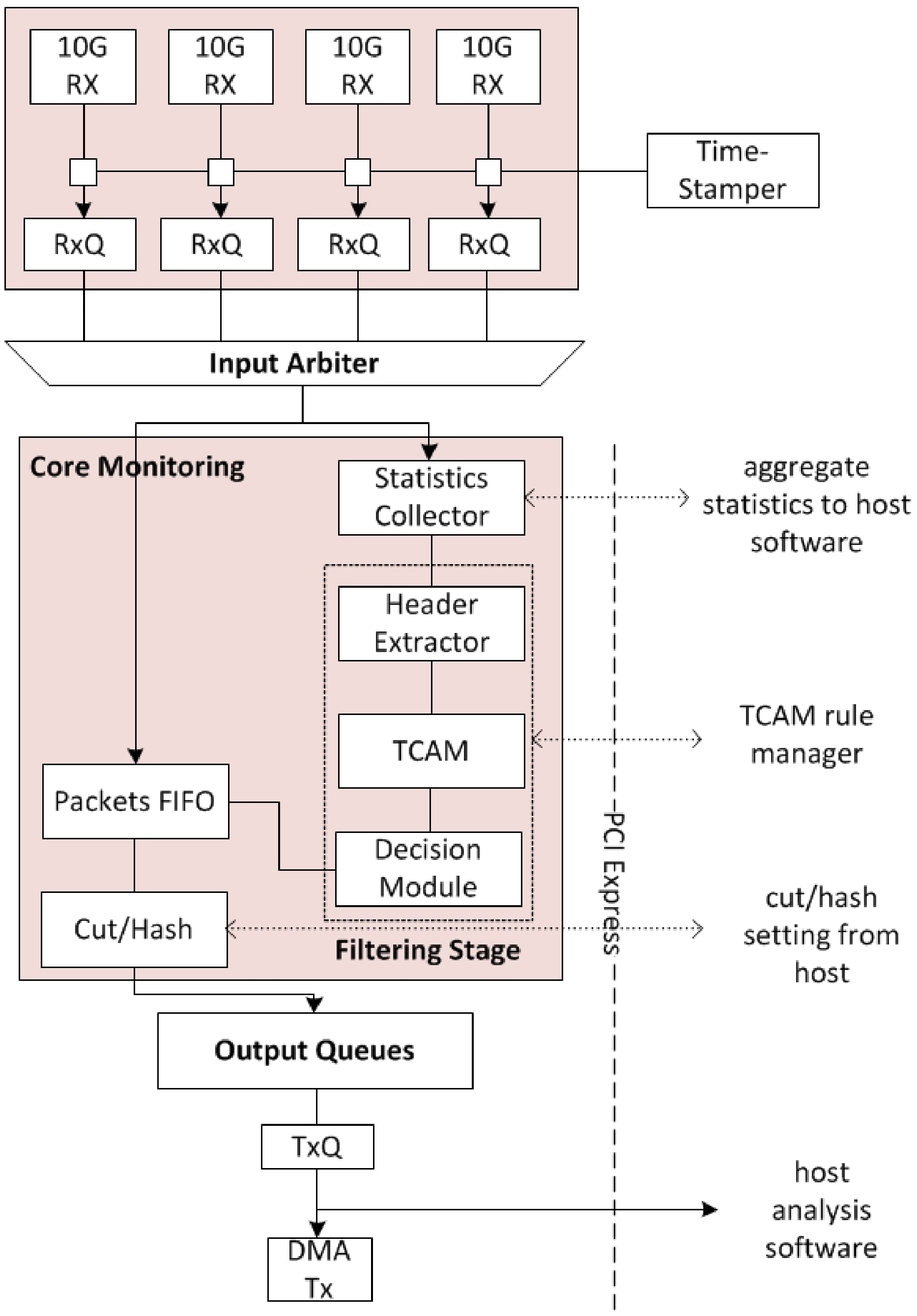


Figure 2.3: Reference pipeline of the OSNT project

3 Measuring Network parameters

In the IP network that the philosophy of the best-effort is used, it is important to know the quality of the networks. This information is valuable in order to configure services that are sensible in low packet losses, low one-way delay and low jitter like VoIP services, online games, etc. For the network's measurements performance are used two different methods: the packet generation method and the packet time-stamping method.

The packet generation used involves the injection of probe packets into the network for measuring the statistics results. Specifically, there is a system (for example a computer's network interface) that generates a flow of packets of the same header and payload size, on the one end, while on the other there is a receiver that receives this flow of packets. The generation rate is the maximum bandwidth of the network connection. With this technique the network devices are tested at the highest load. The metrics are calculated by observing the size, the amount and the arrival time of the packets. The network parameters are throughput, packets per second jitter and packet lost. Throughput characterizes the amount of data that the network can transfer per unit of time and is measured in bits per seconds (bps). Jitter or Packet Delay Variation (PDV) is the difference in end-to-end one-way delay between selected packets in a flow with any lost packets being ignored. Due to the devices' performance and the actual state of the network, the statistics can be variable and might not achieve the theoretical limits.

At the packet time-stamping technique, which is the technique that is implemented in this system, a small number of generated back-to-back packets is enough for measuring the network parameters of the network. The generated packets are sent all together at the maximum speed in order not to be separated between them as the figure 3.1 illustrates. In the case where the packets traverse a bottleneck link, a space will be added between each pair of packets. Such space will be determined by the bit rate of the bottleneck link. The measured quality parameters are the bandwidth, one-way delay, packet losses and jitter. The calculation of these parameters are based on the received time and the time-stamp that each received packet. For the calculation of the instantaneous available bandwidth (R) for each pair of packets in the train will be given by the length in bits of a packet (L), divided by the difference between the arrival times of both packets as the equation 3.1 describes.

$$R_{(i+1)} = \frac{L}{Receiver\ Timestamp_{(i+1)} - Receiver\ Timestamp_{(i)}} \tag{3.1}$$

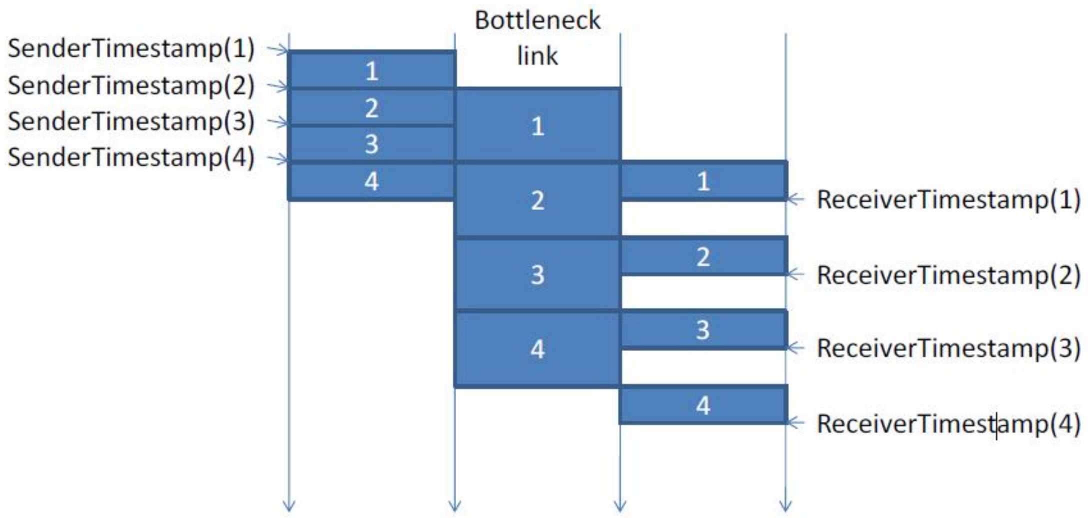


Figure 3.1: Time stamping technique

Based on this idea the mean available bandwidth for all packets the train is calculated.

The receiving test unit compares the send time stamp of the test packet with the time of its reference clock(receive time). This difference is the one-way delay of the link. To measure the jitter, the variation in one-way delay between subsequent packets is calculated. For the packet losses calculation the receiving sequence number order of each packet is verified.

Two key factors affect the resolution and accuracy of these network parameters: synchronization error between the test units' reference clocks, and the internal inaccuracy latency error of the measurement device itself. Both of these must be minimized to provide a meaningful one-way delay measurement.

The most reliable and accurate approach is the GPS Receiver-Based Synchronization. This approach embeds a networking timing source directly into one of the measurement units by using a reliable system of GPS. GPS eliminates the inaccuracies. This method matches the sync path to the service transport path under test, while also eliminating delays related to accessing a remote time server like in the case of NTP-server synchronization [15].

3.1 GPS synchronized frequency/time source

For the accurate time-stamping of the packets a GPS system is used. Global Positioning System (GPS) is a global satellite network consisting of at least 24 satellites maintained by the US Department of Defense (DoD). The present system provides full 24-hour service for high precision time broadcast and two-dimensional navigation. In operation, each of the satellites is continually broadcasting its own position and its own very precise time. Only a small adjustment -the periodic addition of leap seconds- is needed to make this time equivalent to Universal Time Coordinated (UTC). Most GPS receivers make this adjustment automatically, so the time reported to the user is UTC. The satellites broadcast regularly recalculated and updated ephemerides, so their position in space can be accurately calculated as well. Using this information, a ground - based receiver can accurately track time and triangulate its position provided that there are at least four satellites in view. Applications of GPS-based frequency and time sources include network communications, production test and calibration and electric power distribution.

A GPS-based time source, or GPS clock, addresses the foregoing key concerns of communications providers. Frequency, accuracy and stability are similar to those obtainable from atomic frequency standards. The GPS system offers time stability to within 300 ns. Typical pulse-to-pulse jitter in GPS timing receivers however, is 40 to 60 ns. In the design of GPS receivers, emphasis has conventionally been placed on producing a time output (1 pulse per second, or 1 pps) having a short-term average that is very accurate in relation to GPS time. Pulse-to-pulse jitter has typically not been an overriding concern [16, 17].

In any case, GPS clock synchronization eliminates the need for manual clock setting (an error-prone process) to establish traceability to national and international standards so various events can be correlated even when they are time-stamped by different clocks. The benefits are numerous and include: legally validated time stamps, regulatory compliance, secure networking, and operational efficiency.

There are lots of manufactures with interests of time Synchronization to GPS [17, 18]. For this work it is used the EVK-5T model of ublox. It is low cost GPS system but it provide us the currency that this project demands.

4
Implementation

The hardware implementation includes the modules and their functionality of the hardware and the software program that analyse the received packets and calculates the network parameters in off-line mode. In this chapter also it is illustrated the registers of the cards that are accessed. In addition the full interconnection of the hardware modules is shown. The verification of the correct project functionality is presented.

4.1 Packet Generator Module

The Packet Generator module's purpose is to generate dynamically UDP packets and sent them to the network. The module is parameterized through the register system by writing the appropriate register of the card. The access of the registers is done through the PCIe, DMA system and driver. The fields of the UDP packets are filled according to the user selections and the IEEE standards as the appendixA.1 explains.

The Packet Generator module is constructed by 3 sub-modules as the figure 4.1 illustrates: ipif_regs module, rater module and generator module. The ipif_reg module is responsible for communicating with the host through an AXI LITE interface and gives the parameters to the rater and generator modules. The rater module is responsible for the frequency of generation packets. Last, the generator constructs and sends the packets through an AXI STREAM interface.

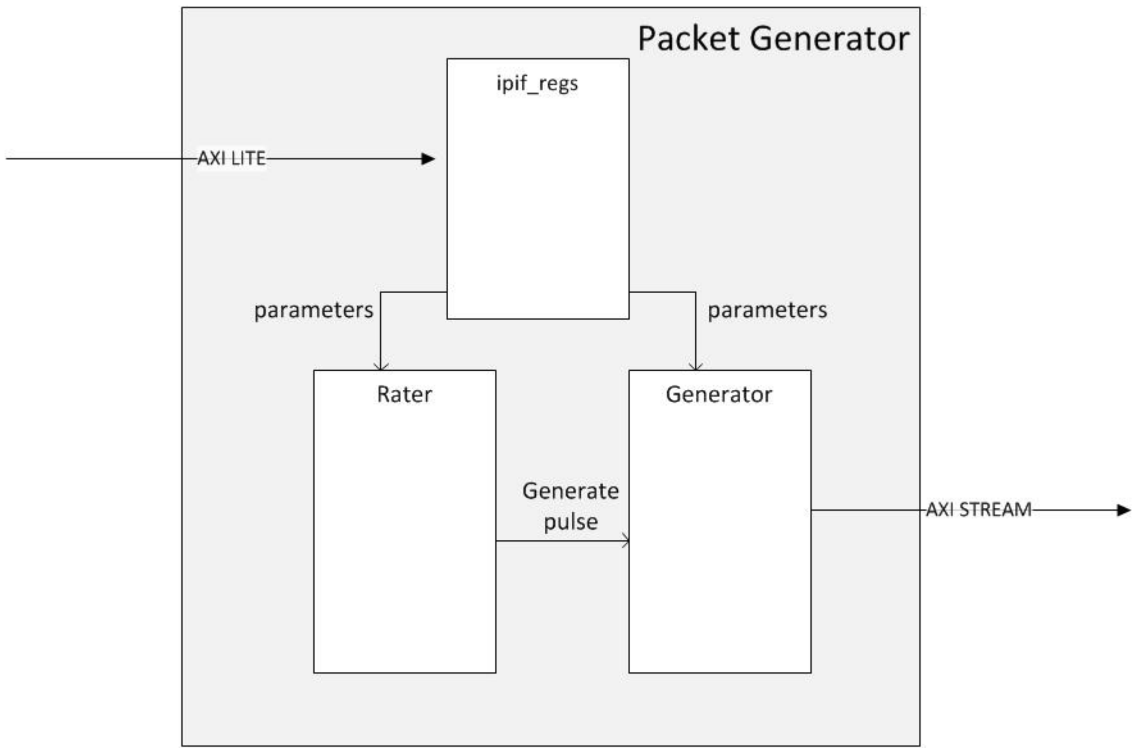


Figure 4.1: Packet Generation module

4.1.1 Rate module

Rate is a hardware module implemented in Verilog. It is responsible for the periodical time announcement of the packet generation. The rate module is parameterized by the host in order to adjust the period of the packet generation and as a result the generation speed. Rater module has two 32-bit register that adjusts the generation period. The first register called limit is set up by the register system. The second register called counter is increased by +one every positive edge of the clock. When the counter reaches the same value as the limit it reset to zero and the one-bit generate_pulse wire is send one pulse that indicates the start of a new generation packet as the figure 4.2 illustrates. Each positive edge of the clock is 6.25ns as the 160MHz reference clock defines.

This module is independent from the state of the generator module. This module depends only on the reference clock of the card. The reason is that the packet generation period time should always be constant and independence of the next connected module state that is the Generator module . Whenever it is time to send a packet, the Rater indicates this by raising the generate_pulse wire for one clock cycle.

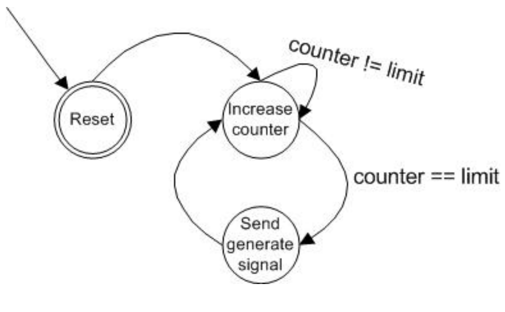


Figure 4.2: Rater module's FSM

4.1.2 Generator Module

Verilog module of Generator constructs dynamically and sends UDP packets to the next module. The headers of the UDP packets are filled with the selections of the user and the IEEE standards as the appendixA.1 explains. The user preferences reach to the Generator module through the register system that the standard library of the NetFPGA provides.

State Machine

The Generator module has one State Machine. The execution of the states constructs dynamically packets according to the standards and the user choices and sends the packet to the next module at chunks of m_axis_tdata weidth (256 bits). The state machine consists from six states as the figure 4.3 illustrates: Wait, Prepare, Send_Header, Send_Header_Payload, Send_Payload, End_Packet.

The state machine is initialized at the Wait state after the first hardware reset. While there is not generate pulse indication the state machine remains at same state. When a new generation pulse is arrived the state is changed at Prepare.

At the Prepare state, the new packet is constructed depending on the parameters that are chosen. The registers that represent the three network layers, MAC, IP, and UDP are filled according with the parameters. This state lasts one clock.

The next state after Prepare state is Send Header where the first 256 bits of the packet are sent to the next module, filling all the wires of the AXI_STREAM protocol. It is standard that the first 256 bits of the packets is part of the header. The Send_Header state lasts one cycle before the state machine changes to Send_Header_Payload state.

Send_Header_Payload state is responsible to send the next 256 bits of the packet. As the packet header has standard size of 336 bits, at this state it will be send 80 bits of the header and 176 of the payload if there are. In case that there is not more payload bites to be send the state changes to End_Packet. If there are more payload bites to be send the next state changes to Send_Payload.

The bits of the payload are sent at chunks of 256 bits at the Send_Payload state. When there is not more payload to be send the state changes to End_Payload by the indication of the one-bit wire end_of_packet. When there are no more payload bits of be send the state of the state machine changes from End_Packet to Wait.

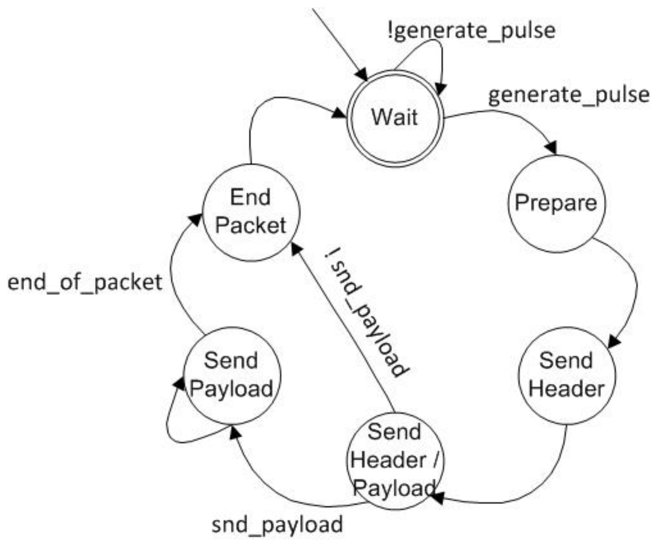


Figure 4.3: Generator module's FSM

Module Functionality

There are three registers that represent the three header of the packet and two more that are for the time-stamp and the sequence number of the packet as the table 4.1 shows. The size of the registers depends on the size of the header that the IEEE standard defines. The fields of the packets are filled with fixed values or with user values or with calculated values that the hardware does. The tables 4.1, 4.2, 4.3 and 4.4 illustrates the fields' values of each header that are filled.

Table 4.1: Packet Headers' name and size

Reg/Wire	Size in Bytes	Name
reg	14	mac_header
reg	20	ip_header
reg	8	udp_header
wire	64	timestamp
reg	32	Sequence_number

Table 4.2: Fields and values of MAC header

MAC Header Values	
Field	**Value**
mac_destination	User choice
mac_source	User choice
mac_type	0x800

Table 4.3: Fields and values of IP header

IP Header Values	
Field	**Value**
Version	4
IHL	5
Differentiated Services	User choice
Tolat length	Calculated value
Identification	Calculated value
flags	0
Fragment Offset	0
TTL	255
Protocol	17
Header Checksum	Calculated value
Source IP address	User choice
Destination IP address	User choice

Table 4.4: Fields and values of UDP header

UDP Header Values	
Field	**Value**
Sourch UDP address	User choice
Destination UDP address	User choice
Length	Calculated value
Checksum	0

The values of the user are reaching to the Generator module through the register system. They are 32-bits input wires that are connected with the ipif_regs module. All the input parameters wires have the size of the fields that the IEEE standards define.

The values that should be calculated by the hardware are the length of the IP header and the length of the UDP header. For the length of the IP header it is calculated the bytes of the IP, UDP header, plus the payload size in bytes. Finally, as the size of the header is standard the length calculation is:

$$length = 28 + payload\_size\_bytes[cur\_queue\_stored] \tag{4.1}$$

The calculation of the checksum demands two cycles of calculations. At the beginning, the checksum field is initialised a zero.In the first cycle the header of the IP is cut in chunks of 16 bits and they are summed into the 19-bits temp register.

$$\begin{aligned} temp[0:18] = ((&ip\_header[0:15] + ip\_header[16:31])+ \\ &(ip\_header[32:47] + ip\_header[48:63]))+ \\ &((ip\_header[49:79] + ip\_header[80:95])+ \\ &(ip\_header[96:111] + ip\_header[112:127]))+ \\ &(ip\_header[128:143] + ip\_header[144:159]) \end{aligned} \tag{4.2}$$

In the second cycle, if the temp register produced a number bigger than 16 bits, the extra bits are summed up to a 16-bits result (sum register) which is then subtracted out of 0xFFFF.

$$sum[0:15] = temp[3:18] + temp[0:2] \tag{4.3}$$

$$ip\_header[80:95] = 0xFFFF - sum[0:15] \tag{4.4}$$

For the length of the UDP header it is calculated the bytes of the UDP header, plus the payload size in bytes. Finally, as the size of the header is standard the length calculation is:

$$length = 8 + payload\_size\_bytes[cur\_queue\_stored] \tag{4.5}$$

After the UDP header it is added the payload. The payload begins always with the same 32-bits fields of the time-stamping. The time-stamp register consists of two 32-bits parts: the seconds and the nanoseconds. The register size it is chosen so bin in order to be able to represent the actual second. The register size of the nanosecond is big enough in order to have precision of 6 nanoseconds. This information comes as input from the module of the GPS that controls these registers. The serial number is register that indicates the number of the sending packet. Every time a packet is sent, it increases +1. The serial number is used by the receiver in order to identify lost packets. Each time a new raw of packets is send this counter is reset to zero.

These two fields are included in the RTP header. However the RTP header is not used in this system. First, the RTP header size is much bigger than manual mini header that is proposed. This means that there is less overhead at each send packet. In addition, the RTP header provides less accuracy at the type stamp as the RTP time stamp field is 32 bit when our proposal is 62. Therefore RTP header is not adequate for network measurements with high precision with the use of time-stamps.

The NetFPGA process the packet data of the network with difference endianness that the data are sent in the network. Because of this, the send packet bytes should be sent inversely as the figure 4.4 explains.

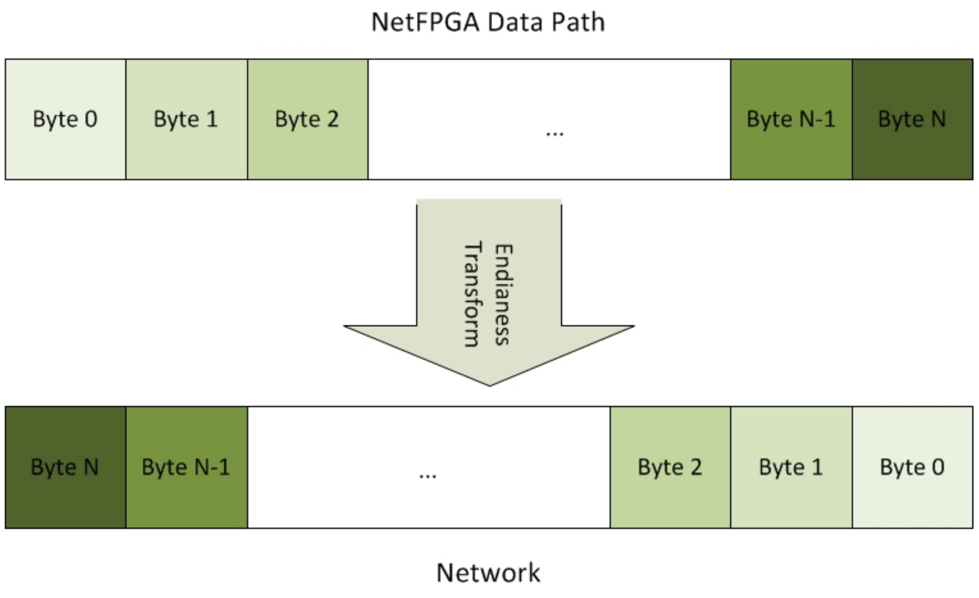
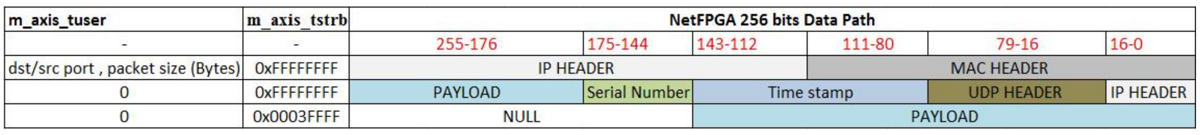


Figure 4.4: Endianess

The size of the heads with our "new header" is always standard. Their size is 54 bytes or 432 bits. Consequently, the first packet data transmission is the only header as at each transmission it is sent 256 bytes. The value of the m_axis_tstrb wire of the first transmission is 0xFFFFFFFF as all the bytes are valid. In addition, in the first transmission the m_axis_tuser that indicates the packet size in bytes and the source/destination port is filled.



m_axis_tuser	m_axis_tstrb	NetFPGA 256 bits Data Path					
-	-	255-176	175-144	143-112	111-80	79-16	16-0
dst/src port , packet size (Bytes)	0xFFFFFFFF	IP HEADER			MAC HEADER		
0	0xFFFFFFFF	PAYLOAD	Serial Number	Time stamp		UDP HEADER	IP HEADER
0	0x0003FFFF	NULL		PAYLOAD			

Figure 4.5: AXI Stream registers' values by sending a UDP packet of 32 bytes payload

The second transition as the state machine at figure 4.5 illustrates, header and payload data are sent. The first 176 bites of the transmission are the header of the packet plus the time-stamp and the serial number and the 80 more bytes of the transmission are payload if there are. Depending if there is enough data to be send or not the next transmission can be either payload data or the end of the transmission. The valid bytes of the transition are indicated by the value of the m_axis_tstrb wire as the table 4.5 illustrates. For each valid data bit one bit of the of m_axis_tstrb turns to one. The value of the m_axis_tuser wire is zero as it is important only at the first transition. The data value is always a fixed value and the size of the value depends on the users choices.

If there is more payload information, at each clock 256 bits of the payload is sent with m_axis_tstrb value of 0xFFFFFFFF. For the last transmission of the data the value of the m_axis_tstrb indicates the valid data like at the earlier occasions.

Table 4.5: Value of m_axis_tstrb

Value of m_axis_tstrb	**Data Valid Bytes**
0x0	0
0x1	1
0x3	2
0x7	3
...	...
0x7FFFFFFF	255
0xFFFFFFFF	256

4.1.3 Register System

The register system that is implemented on the hardware of FPGA is the way that the host communicates with the NetFPGA card. The register system modules are included in the standard library of the NetFPGA and collate and generate addresses for all the registers and memories in a NetFPGA project. Each module defines the memory size and the address that needs for the registers at the mpd file of its data folder. The communication is achieved through the DMA mechanism and the PCIe interface. The driver of the NetFPGA at the software level receives and sends the request from and to the card. The user has simple functions for reading and writing the address of the register system. The tables 4.6 and 4.7 illustrate the name, the use and the length of the registers for the Packet Generator module.

Table 4.6: Packet Generator Module's registers

Packet Geretor Module		
Register Name	**Description**	**Size in bits**
payload_size_bytes	Payload size in bytes.	32
dstip	Destination IP address by decadal representation.	32
srcip	Source IP address by decadal representation.	32
dstport	Destination port (field of transport layer).	32
srcport	Source port (field of transport layer).	32
dstmac_high	High 16 bits of destination MAC address.	32
dstmac_low	Low 32 bits of destination MAC address.	32
srcmac_high	High 16 bits of source MAC address.	32
srcmac_low	Low 32 bits of sourch MAC address.	32
num_packets	Number of packets to be generated.	32
num_packets_generated (output)	Generated packets number.	32

Table 4.7: Rater Module's registers

Rater Module		
Register Name	**Description**	**Size in bits**
clk_limit	Packet generator flow rate limit.	32
packets_generated (output)	Number of generated packets of the Rater module.	32

4.1.4 Software Analysis

The hardware receiver marks them the arrival time when the packets arrived. The time-stamp precision is at the scale of nanosecond. The packet is sent through the PCIe and the DMA to the host memory and when the user needs to store them, executes a user level program that copies the packet from the memory to the hard disk. The saved format is pcap. As the standard libpcap library offers precision at the scale of ms for the saved packets, the new unreleased version of the library 1.6.2 should be compiled and used. With this change the packets are saved with precision of nanosecond that it is demanded for network measurement at 10Gbps.

The analysis program that is implemented reads the saved pcap files and analyze them in order to obtain the network measuring parameters of the network. The analysis program uses the libpcap library as a result it should be compiled with the new libpcap library 1.6.2.

As input argument the program expects pcaps file. All the packets are captured for all the pcap files. The time stamps and the serial number of each packet of the pcap file are analyzed. The arrival time-stamps of each packet is used for the calculation of the throughput. The information of the arrival time-stamp is stored at the meta-data of the header of the packet as the figure 4.6 illustrates. The instant throughput is calculated with the arrival time subtraction of two consecutive packets as the follow equation illustrates. Respectively the average throughput is calculated.

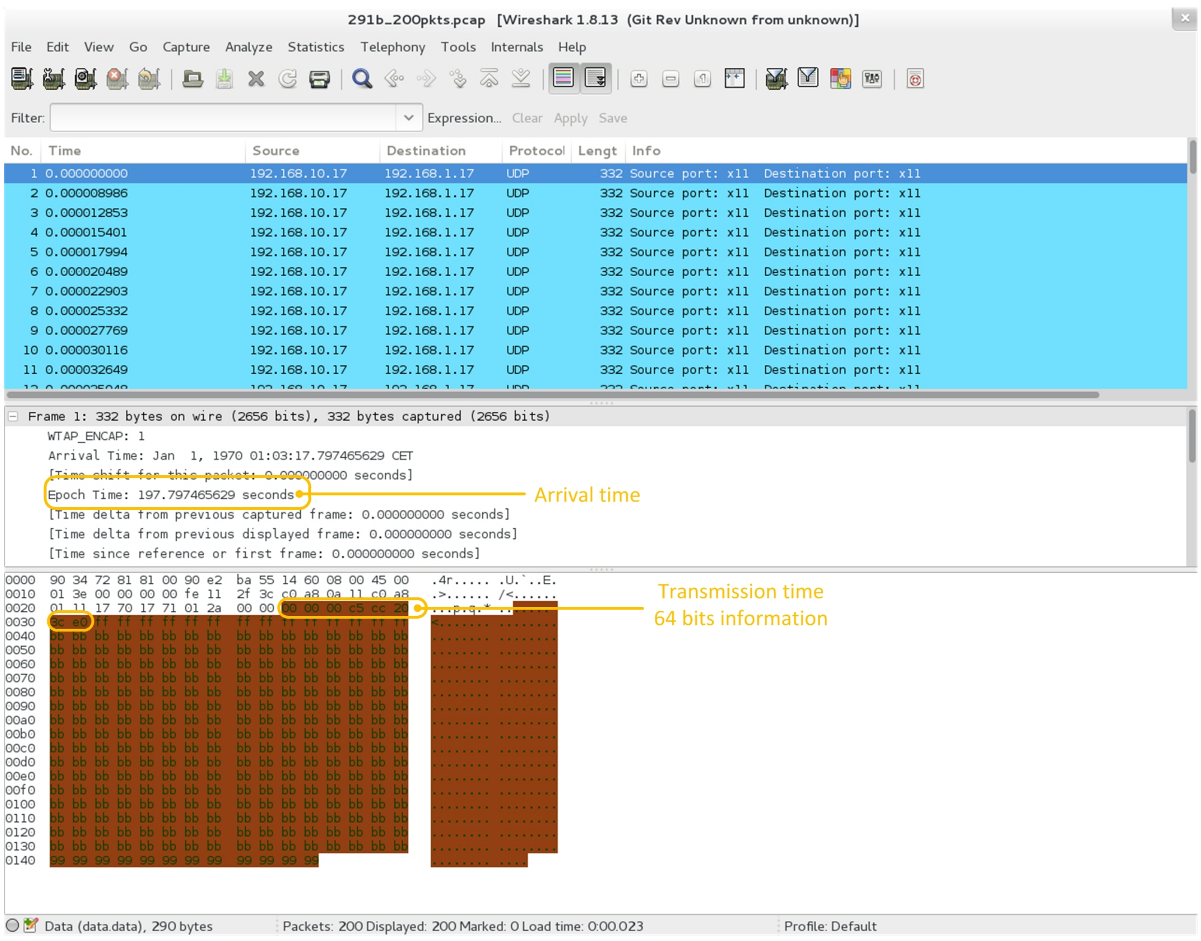
$$Throughput = \frac{L}{arrival\ time_{(i+1)} - arrival\ time_{(i)}} \tag{4.6}$$

The actual jitter is calculated with the arrival time of the three last consecutive packets as the follow equation shows.

$$Jitter = ||arrival\ time_{(i+2)} - arrival\ time_{(i+1)}| - |arrival\ time_{(i+1)} - arrival\ time_{(i)}|| \quad (4.7)$$

Respectively the average jitter is calculated. The one-way delay is calculated with the arrival time and the transition time of all the packets as the follow equation shows. Respectively the average one-way delay is calculated.

$$OnewayDelay = arrival\ time_{(i)} - transmition\ time_{(i)} \quad (4.8)$$


291b_200pkts.pcap [Wireshark 1.8.13 (Git Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save
Arrival time
Transmission time
64 bits information

```
No.  Time          Source          Destination    Protocol Lengt Info
  1 0.000000000    192.168.10.17   192.168.1.17   UDP        332 Source port: x11  Destination port: x11
  2 0.000008986    192.168.10.17   192.168.1.17   UDP        332 Source port: x11  Destination port: x11
  3 0.000012853    192.168.10.17   192.168.1.17   UDP        332 Source port: x11  Destination port: x11
  4 0.000015401    192.168.10.17   192.168.1.17   UDP        332 Source port: x11  Destination port: x11
  5 0.000017994    192.168.10.17   192.168.1.17   UDP        332 Source port: x11  Destination port: x11
  6 0.000020489    192.168.10.17   192.168.1.17   UDP        332 Source port: x11  Destination port: x11
  7 0.000022903    192.168.10.17   192.168.1.17   UDP        332 Source port: x11  Destination port: x11
  8 0.000025332    192.168.10.17   192.168.1.17   UDP        332 Source port: x11  Destination port: x11
  9 0.000027769    192.168.10.17   192.168.1.17   UDP        332 Source port: x11  Destination port: x11
 10 0.000030116    192.168.10.17   192.168.1.17   UDP        332 Source port: x11  Destination port: x11
 11 0.000032649    192.168.10.17   192.168.1.17   UDP        332 Source port: x11  Destination port: x11
```

```
Frame 1: 332 bytes on wire (2656 bits), 332 bytes captured (2656 bits)
    WTAP_ENCAP: 1
    Arrival Time: Jan  1, 1970 01:03:17.797465629 CET
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 197.797465629 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
```

```
0000  90 34 72 81 81 00 90 e2  ba 55 14 60 08 00 45 00   .4r..... .U.`..E.
0010  01 3e 00 00 00 00 fe 11  2f 3c c0 a8 0a 11 c0 a8   .>...... /<......
0020  01 11 17 70 17 71 01 2a  00 00 00 00 00 c5 cc 20   ...p.q.* ........
0030  3c e0 ff ff ff ff ff ff  ff ff ff ff ff ff ff ff   <....... ........
0040  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
0050  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
0060  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
0070  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
0080  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
0090  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
00a0  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
00b0  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
00c0  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
00d0  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
00e0  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
00f0  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
0100  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
0110  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
0120  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
0130  bb bb bb bb bb bb bb bb  bb bb bb bb bb bb bb bb   ........ ........
0140  99 99 99 99 99 99 99 99  99 99 99 99               ........ ....
```

```
Data (data.data), 290 bytes    Packets: 200 Displayed: 200 Marked: 0 Load time: 0:00.023    Profile: Default
```

Figure 4.6: Wireshark capture of generated packet

4.2 System Integration

The Verilog module of the Packet generator is integrated with the OSNT traffic monitor of the OSNT project. The Packet generator module is connected through the AXI STEAM interface with the first Tx Output Queue of the NetFPGA as the figure 4.7 illustrates. The module is connected with the AXI LITE bus for the communication with the host through the PCIe in order to set the parameters of the module (number of generated packets, size of packets, etc.). The time-stamp input of the module is connected with the wire of the time-stamp that the OSNT traffic monitor provides.

The Packet generator module is set with the parameters of the user by the register system and starts to generate and send packets of the same delay and the same packet size to the network. The generated packets have the time-stamp GPS that provides the OSNT traffic monitor project.

The time-stamped packets traverse the network under measurement and arrive at the interfaces of the NetFPGA. Then the OSNT receiver receives the packets and marks them with the GPS active value as soon as possible. The time-stamped received packets pass the monitor core where they are filtered, cut and forwarded to the DMA queue. When there are packets at the DMA queues the DMA engine sends the packets through the PCIe to the host memory. A user level program is responsible to copy the data from the host's memory to the hard disk at pcac form. Our software reads the saved pcaps files and makes an offline analysis of the flow. The network parameters are calculated so the network performance is specified.

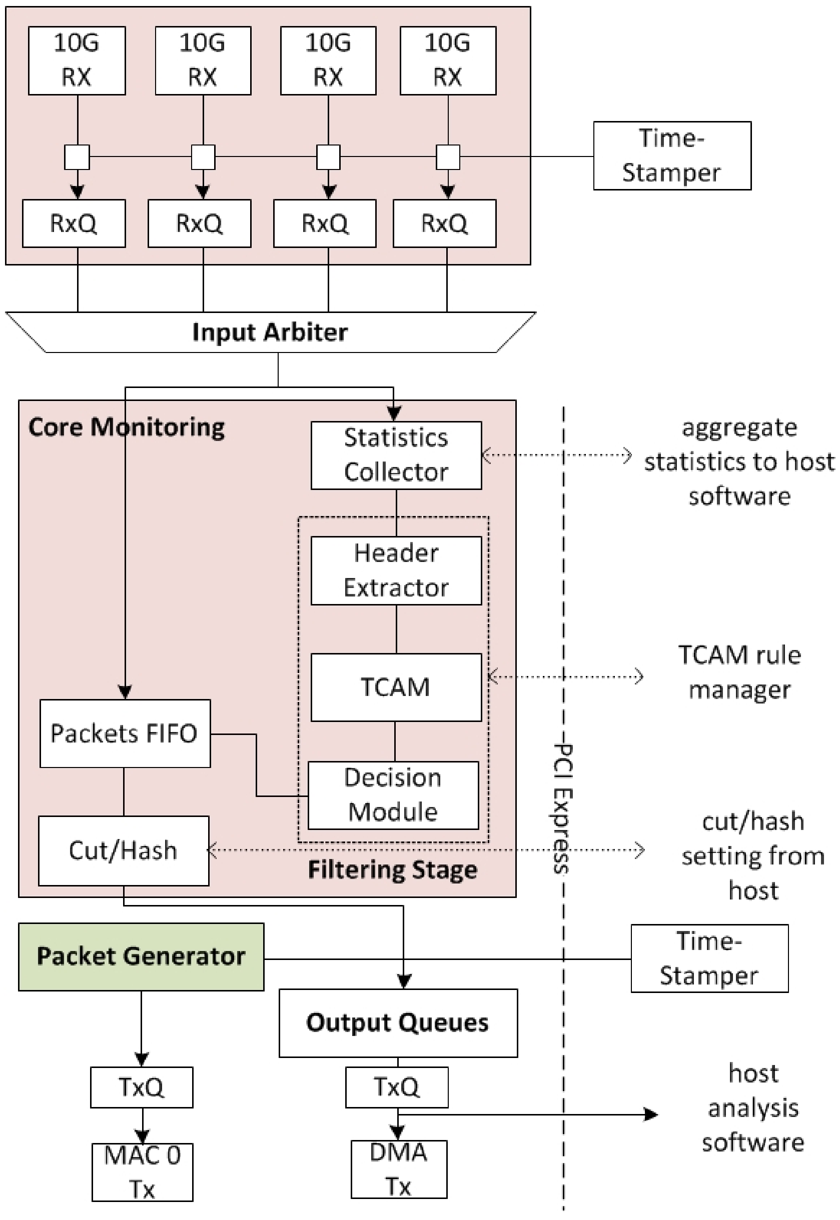
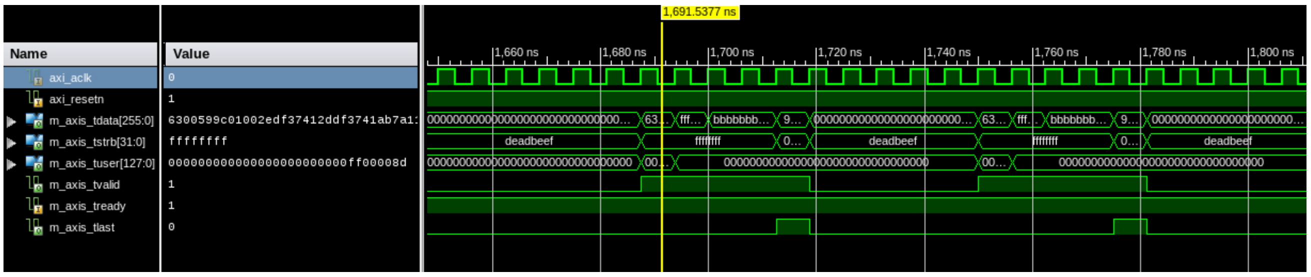


Figure 4.7: user_data_path_new

4.3 System verification

The verification of the system is important as it assurances the capabilities of the project. The verification part includes the simulation part and the real time execution of the project.

The simulation part includes the behavioural simulation of the module and the entire system of the hardware part in order to verify that they work properly and the wires' values are the expectable. In addition, wrong connection or unconnected wires are checked at this step. A python file which is responsible for generating the Verilog testbench is written. The python file initializes the network interfaces of the simulated system and writes the appropriate values of each module registers. All the Packet Generator module's registers are set for testing the functionality of the module. The registers and the wires of the module are observed and especially the wires of the AXI STEAM protocol in order to verify that the restrictions are completed as the figure 4.8 shows. The ISim simulation release version 13.4 is used for the simulations. All the wires are connected properly and their values are the expected.



Name	Value
axi_aclk	0
axi_resetn	1
m_axis_tdata[255:0]	6300599c01002edf37412ddf3741ab7a1:
m_axis_tstrb[31:0]	ffffffff
m_axis_tuser[127:0]	00000000000000000000000ff00008d
m_axis_tvalid	1
m_axis_tready	1
m_axis_tlast	0

Figure 4.8: simulation

The aim of the first basic real experiment is to show that the card and our system can measure a 10 Gbps network with high precision using the GPS for time-stamping the packets. The interface nf0 is connected with the interface nf1 directly with an optical fiber. A flow of 200 packets of 1472 bytes of payload are generated in order to verify the ability of the system to measure the theoretical limits. The packet are generated and sent from the nf0 interface and are received from the nf1 interface as the figure 5.4 illustrates. The off-line analysis demostrates at the table 4.8.

The results reveal that the system can measure the bandwidth of a Gigabit connection with high precision. The measured throughput is really close to the theoretical throughput limits. The jitter is close to the period of the clock which is 6.25ns. One-way delay is at micro second scale. 15,50% of the generated packets are not stored by the driver even though all the packets are received from the hardware. Even though, cards of PCIe generation 2 of 8 lanes gives 40Gbps throughout, the DMA engine of the NetFPGA card is not able to send packets to the host at this high rate. This concludes that because of the hardware implementations there are some limitation but for throughput measurements with generated packets of payload 1472 bytes high precision is provided.

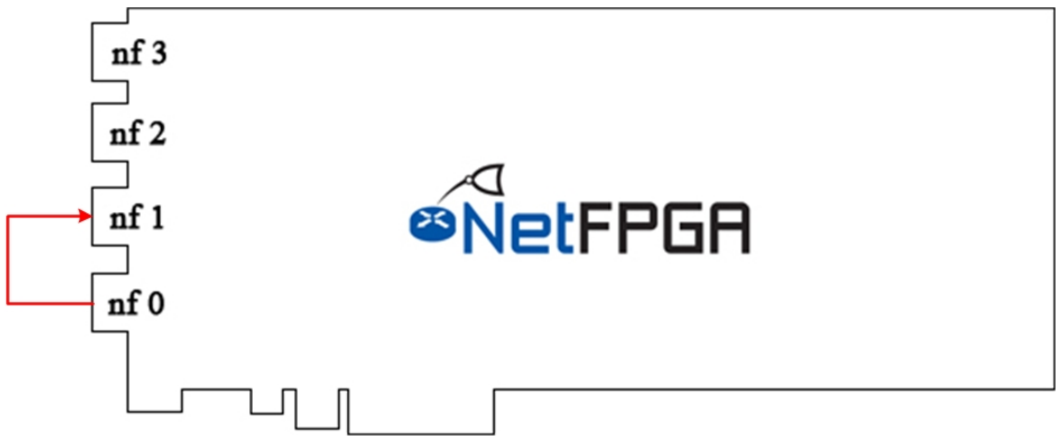


Figure 4.9: Experiment's results

Table 4.8: Rater Module's registers

Payload Size	**Theoretical Throughput**	**Measured Throughput**	**Difference**	**Jitter**	**One-Way Delay**	**Packet Loss**
Bytes	Bps	Bps	%	ns	us	%
1.472	9.843.953.186	9.788.530.843	-0,56	8,631	2,058	15,50

5 Evaluation

5.1 Theoretical limits

Maximum theoretical throughput of 802.03 and 802.11g standards are calculated in order to be used as reference values for the experiments' execution over Ethernet. The payload limit size is between 18 and 1472 bytes as the MAC layer segment defines [**?**].

The maximum theoretical limits of a 1 Gbps throughput connection for 802.03 are calculated with the following formula:

$$pps = \frac{Throughput}{Packet\ Size} = \frac{Throughput}{MAC\ header + IP\ header + UDP\ header + payload} \tag{5.1}$$

$$Throughput = (Frame\ size - 24) * pps \tag{5.2}$$

For the measurements the packet size used is outlined in the table 5.1:

Table 5.1: Pps and Throughput theoretical limits vs Payload Size

Packet Size	Packet Useful data	Payload Size	Max pps	Max Throughput
Bytes	Bytes	Bytes		Bits per second
84	60	18	14.880.950	7.142.857.143
300	276	234	4.166.667	9.200.000.000
1000	976	934	1.250.000	9.760.000.000
1538	1514	1472	812.744	9.843.953.186

The bigger the payload size, the better the network's sources are utilised. However, the theoretical total throughput is not achieved because of the packet's overhead, preamble, start of frame delimiter, inter-frame gap and frame check sequence with total size 24 bytes. On the other hand, with smaller payload size, smaller throughput is achieved because of the useful information's fraction and the extra payload overhead. Fig. 5.2 shows the 802.03 link's theoretical throughput and the fig. 5.1 shows the theoretical 802.03 link's pps.

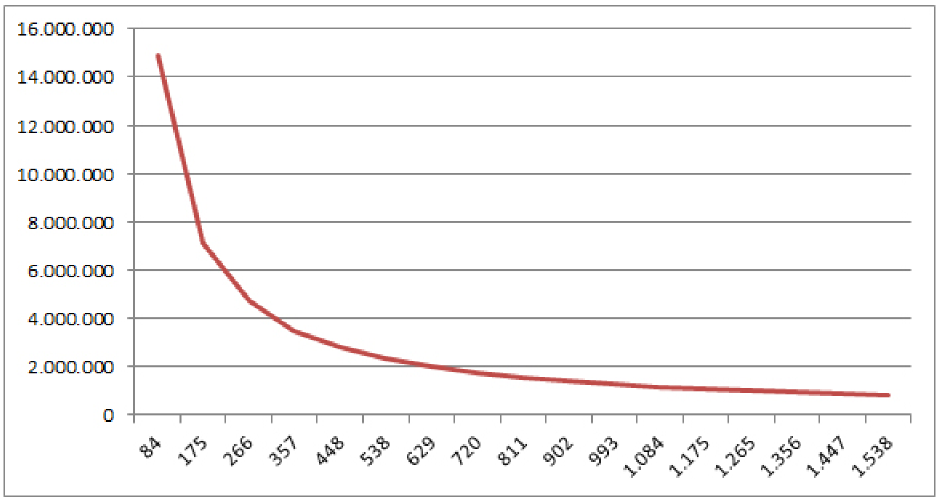


Figure 5.1: 802.03 flows' theoretical packets per second limits

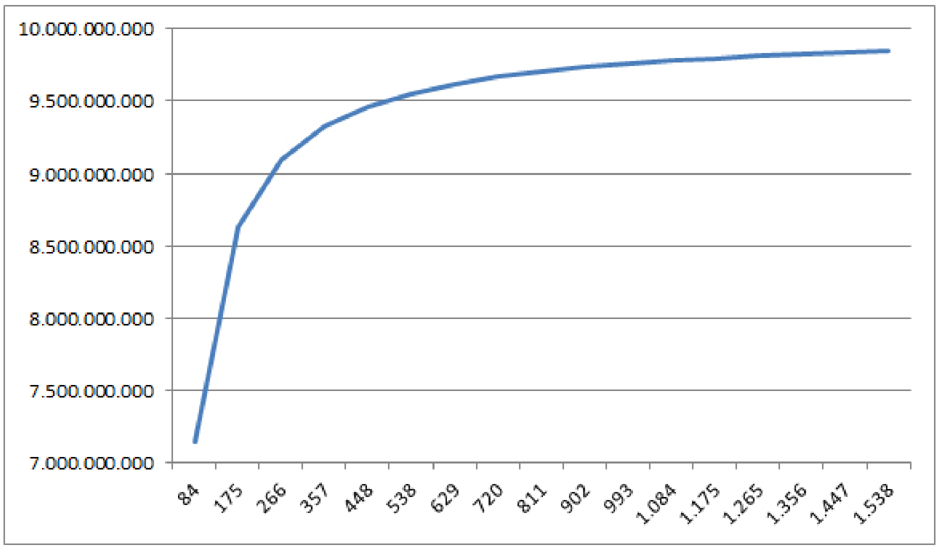


Figure 5.2: 802.03 flows' theoretical throughput limits

5.2 Testing equipment

For the experiments it is used two computers. Even though the hardware of the NetFPGA has a high standard performance, the computer that hosts the card affects the performance of the experiments as the software part of the computer is used in order to store the packets. In addition, in some of the following experiments it is used a second computer and a network card for simulation of a real network.

The first computer is used as a host for the NetFPGA card. Through this computer the card is set in order to generate the packets that will measure the network. The NetFPGA is used as a network receiver for capturing the packets as well. The performance of the computer affects only at the moment of the packet storing at the disk.

The second computer is used as a packet repeater. The computer has a Gigabit network card with two interfaces. The packets that are entering from the first interface are propagated to the second interface. In software level delay and packet losses is added to the generated flows. The technical features of the two computers are presented at the table 5.2.

Table 5.2: Computer specifications

	NetFPGA computer	**Virtual LAN computer**
CPU	Intel(R) Xeon(R) CPU E5-1620 0 @ 3.60GHz	Intel(R) Core(TM) i7 CPU 920 @2.67GHz x8
RAM	16 GB	6 GB
Network card	NetFPGA-10G Xilinx Virtex-5 VTX240T	Intel Corporation 82599EB 10-Gigabit SFI/SFP+
O.S	Fedora 14	Ubundu 12.04
Kernel	2.6.35.14-106.fc14.x85_64	3.8.0-35 generic

5.3 Experiments

5.3.1 Experiment 1: Theoretical Limits of 10Gbps Link

This experiment is the most important as it proves the capability of the system to generate packets with high precision at low jitter. The aim of this experiment is to measure a 10Gbps link by generating flows of different packet sizes in order to see the difference between the measurements. The theoretical maximum throughput is changing depending on the packet size as the calculations of the paragraph 5.1 shown. Different measured throughput is expected for each packet size. The NF0 network interface of the NetFPGA 10G is connected with the NF1 as the figure 5.3 which is a 10Gbps link as the interfaces supports link at this speed. 200 packets are generated and are sent from the hardware and are saved from the software at different packet sizes.

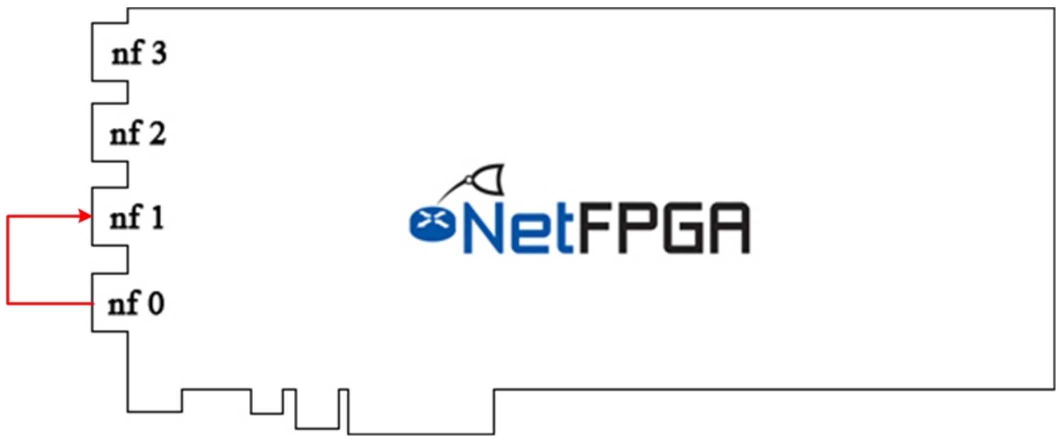


Figure 5.3: Experiment's 1 topology

Table 5.3: Experiment's 1 Network Parameters

Payload	**Jitter**	**One Way delay**	**Packet Loss**	**Packet Loss**
Bytes	**ns**	**us**	**packets**	**%**
18	7.636	1.139	85	0.424
291	7.093	1.923	82	0.41
563	2.734	1.756	48	0.24
745	8.239	1.935	54	0.27
1018	8.960	2.579	66	0.33
1290	7.2931	2.189	43	0.215
1472	8.631	2.057	31	0.155

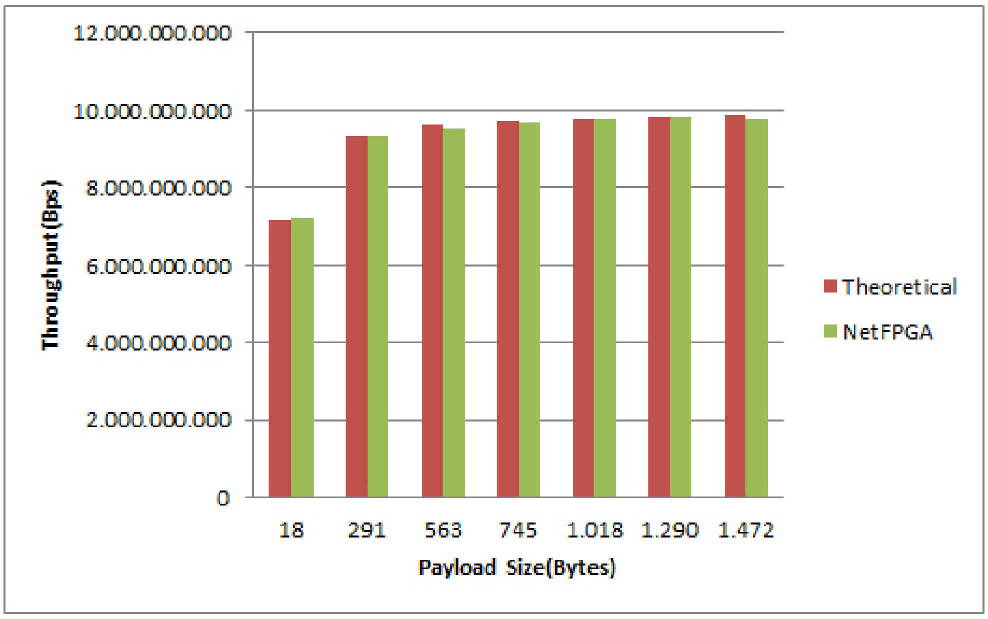


Figure 5.4: Experiment's 1 Throughput vs Packet Size

The results of the table 5.3 shown that the system can measure the 10Gbps link with high precision as for all the packet sizes the difference of the maximum theoretical and the measured throughput is really small(-0,88% at the worst case) . The jitter is at low lever, between 2,734ns and 8,631 ns that is almost the precision that the 160MHz clock of the FPGA provides. The one-way delay is at the scale of us, between 1,140us and 2,579us. The percentage packet loss is really high for all the packet sizes between 15,50% and 45,50%. At hardware level the packets are generated and captured from hardware monitor. This disadvantage is not because of our design but because of the standard library core DMA core of the NetFPGA. The 8 lanes of PCIe of the card have theoretical throughput 40Gbps but in reality this throughput is not achieved. The packet loss phenomenon is examined in detail at the experiment 2.

5.3.2 Experiment 2: Packet Losses

The experiment's 1 results revile the problem of the packet losses because of the limitations of the DMA hardware implementation. At experiment 2 the packet loss phenomenon is examined in detail and it is proposed a possible solution. The topology of the experiment is the same at the experiment 1 (figure 5.3). Flows of the same packet size are tested with different number of generated packets. The aim is to check the limit of the generated packets where there are not packet losses and to check if the number of generated packets affects the packet loss. The payload sized that are used are the smallest (18 Bytes) and the biggest (1472 Bytes) possible that the 802.03 permits.

Table 5.4: Packet Losses of 18 Bytes payload

Number of generated	**Packet Loss**	**Packet Loss**
Packets	**Packets**	**%**
50	0	0
100	0	0
200	85	42.4
1,000	803	80.3
10,000	9680	96.7
50,000	43553	87.1
100,000	88935	88.9

Table 5.5: Packet Losses of 1472 Bytes payload

Number of generated	**Packet Loss**	**Packet Loss**
Packets	**Packets**	**%**
50	0	0
100	6	6
200	31	15.5
1,000	229	22.9
10,000	3630	36.2
50,000	12720	25.4
100,000	25151	25.1

The results of the tables 5.4 and 5.5 show that for both payload there is not packet losses if there are send less than 100 packets. The high packet per second (pps) of the small payload affects to the packet losses. It concludes that the more pps the more packet losses there are. The difference between the two different packet sizes is 49%. For generated number packets bigger than 1000 the percentage losses is almost unchanged for both payload sizes.

Solution

The packet losses, as it is mentioned earlier, are an effect of the DMA implementation at the NetFPGA. In order to prove it and provide a quick solution of this problem there will be stored a certain size of the captured packet. This experiment is the same as the experiment 2 with the difference that the traveled information from the hardware to the software through the PCIe and the DMA mechanism will be less. With this technique the throughput is reduced as at the same time less information is sent. All the received packets are cut at 64 bytes that is enough in order to include the transition time stamp and the sequence number.

Table 5.6: Packet Losses of 82 Bytes payload with 64Byte packet cut

Number of generated	**Packet Loss**	**Packet Loss**
Packets	**Packets**	**%**
50	0	0
100	4	4
200	90	45
1,000	803	80.3
10,000	5424	54.2
50,000	8706	17.4
100,000	43721	43.7

Table 5.7: Packet Losses of 1472 Bytes payload with 64Byte packet cut

Number of generated	**Packet Loss**	**Packet Loss**
Packets	**Packets**	**%**
50	0	0
100	0	0
200	0	0
1,000	0	0
10,000	124	1.24
50,000	128	0.25
100,000	652	0.65

The smaller size of stored packet affects dramatically the packet losses. Less packet losses are observed for both different packet sizes. For the 1472 bytes of sent packet the packet losses are almost 0 as the pps of the generated flow is lower than the flows of the 18 bytes. On the other hand, the flows with the small payload size and 64bytes packet cut still have a big amount of packet losses. This is because the small packet has size 60bytes so with the process of cutting the packet size remains the same size. The experiment reveals that the problem is spotted at the DMA throughput and not at the implementation of our system. Concluding, the problem of the packet losses can be solved either by changing the DMA engine of the NetFPGA that is a really painful and time-cost process or by cutting the received packet and save the possible smallest size.

5.3.3 Experiment 3: Virtual Network

The experiment's goal is to measure the network performance of a virtual network using our system. A computer and a network card with spesification of the table 5.2 are used for the simulation of the virtual LAN. The generated packets are sent from the nf0 interface of the NetFPGA. The packets enter the eth1 interface of the network card and are propagated from the eth2 interface as the figure 5.5 illustrates. Firstly, IP forwarding is enabled on your system in order to forward packets from one interface to the other. Then, forwarding rules are inserted at the iptables. The computer simulates a real network by adding delay, jitters and adjusts the throughput. 200 packets are generated and are sent from the hardware at different packet sizes.

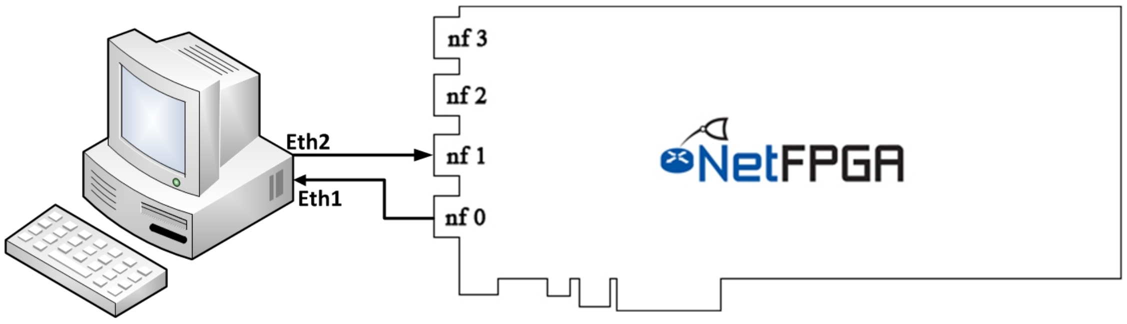


Figure 5.5: Experiment's 3 Topology

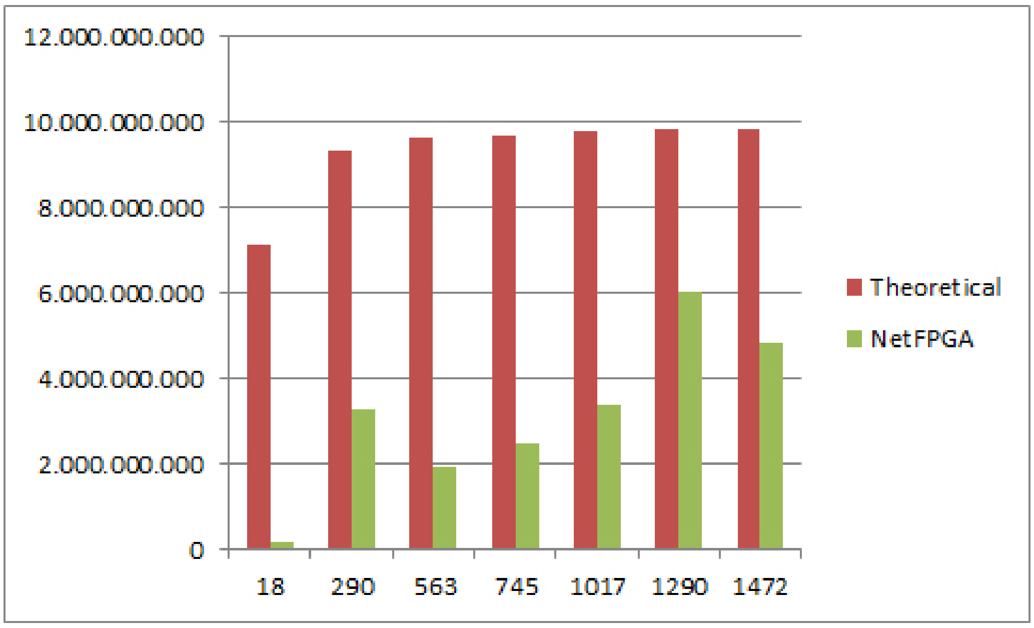


Figure 5.6: Experiment's 3 Throughput vs Packet Size

Table 5.8: Experiment's 3 Network Parameters

Payload	**Jitter**	**One Way delay**	**Packet Loss**	**Packet Loss**
Bytes	**ns**	**us**	**packets**	**%**
18	463.521	290.189	0	0
290	2237.964	361.897	0	0
563	647.565	326.849	0	0
745	655.494	311.315	0	0
1017	727.156	299.271	0	0
1290	583.045	225.149	0	0
1472	677.954	255.081	0	0

The intermediate computer apparently changed the measured throughput as it is much more less than the maximum theoretical limit for all the packet sizes. The differences are between 97.75% and 38.51% as the figure x illustrates. The extra network component is added a high jitter at the generated packets compared with the results of the table x from the experiment 1. The differences are between 456 ns and 2,237ns. The result is logic as the packets from the network should travel all the way up to the kernel in order to retransmit. Retransmission is not done at hardware level but at kernel level. This explanation justifies the extra one way delay of the packet. The extra delay is between 223us and 359us. There are not packets losses at this experiment because the packets are sent with lower throughput back to NetFPGA and with high jitter so probably the NetFPGA buffers are not saturated. Concluding the extra network re-transmitter adds high jitter to the packets, one way delay and reduces the throughput.

5.3.4 Experiment 4: Virtual Network with Delay and Packet Losses

The aim of this experiment is to measure a more realistic network where there are packet losses and delays. Packet losses can be appeared in networks where there works at high picks and the buffers of the routers are getting full very often. The delay is appeared because the packets travel through different traces and the packets should wait at the FIFOs of the routers in order to be routed.

The simulation of these parameters is done at software level using the tc tool. There are done two versions of this experiment. At the first version, the tc tool is set to add 120ms delay and 10% packet losses. At the second version, it is set 520ms delay and 50% packet losses. The topology of the experiment is the same as at the experiment 3 (figure 5.5). 200 packets are generated of different packet size.

Table 5.9: Experiment's 4 Network Parameters version 1

Payload	**Jitter**	**One Way delay**	**Packet Loss**	**Packet Loss**
Bytes	**ns**	**us**	**packets**	**%**
18	1,398.149	520,337.867	99	49.5
290	2,720.999	520,325.076	102	51
563	1,660.937	520,287.676	89	44.5
745	115.851	520,234.750	98	49
1,017	791.107	520,127.161	95	47.49
1,290	1,543.000	520,238.212	106	53
1,472	2082.200	520,223.024	112	56

Table 5.10: Experiment's 4 Network Parameters version 2

Payload	**Jitter**	**One Way delay**	**Packet Loss**	**Packet Loss**
Bytes	**ns**	**us**	**packets**	**%**
18	976.837	120,271.236	27	13.5
290	1,481.220	120,310.471	20	10
563	2,119.191	120,293.261	20	10
745	1,376.653	120,290.819	26	13
1,017	783.122	120,262.754	18	9
1,290	1,195.787	120,242.318	23	11.5
1,472	13.000	120,188.831	56	28

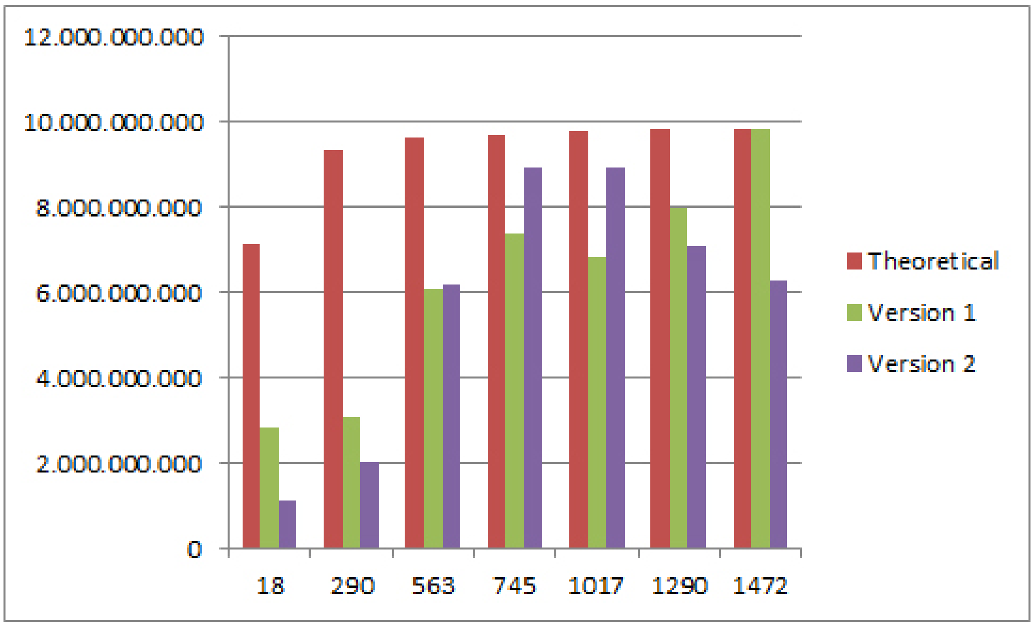


Figure 5.7: Experiment's 4 Throughput vs Packet Size

The new added parameters of the packet loss and the delay of the packets are appeared correctly at our measurements. At the first version the one-way delay is changed approximately to 120,000.000 us and at the second to 520,000.00us which is the parameters of the packet delay of the retransmiter. The percentage packet loss of the measurements is at the first version close to 10% and at the second version close to 50%. The jitter is increased about 1,000ns in all the cases because of the extra calculation overhead of the new rules that are added. This extra calculation overhead affects the calculated throughput of the link. This is the reason why the calculated throughput is less than the theoretical maximum throughput. The processing of the received flows at software level produce this unexpected form of the throughput between the first and the second version as in some cases the version's 1 throughput is greater than the second's and some other cases less.

5.4 Conclusions

The experiments proved that our system can measure 10Gbps networks with high precision. The generated packets have really low jitter and a GPS time-stamping at precision of ns. A limitation at the storing of the packets is observed by measuring of the packet losses, but this problem can be solved easily by cutting the big packet to smaller size. The minimum size that the packets can be cut is the size where the time-stamp and the sequence number are included.

It is also observed that the more software part is involved the worst measurements are observed. At the experiment where it is used an extra computer for simulating a virtual network the calculated throughput was lower than without. The hardware parts of the network (network card, connections) permits to achieve the theoretical limits of the 10Gbps but the software implementation limits the measurements of the throughput. The rest of the network parameters(jitter and one-way delay) are not changed dramatically.

6 Conclusions and future work

Network Measuring is a crucial issue as the performance of the of the networks affects the reputation, the costing and the marketing of the service provides. The measurements should be done with high precision for reliable results. The high rates of the networks make this process more demanding and difficult.

Network measurement with the use of time-stamping is an area with small progress as it is a technique than need high performance system in order to provide high precision measurements. Network measurement systems are implemented over both hardware and software platforms. Software platforms exist both as open-source and free-ware, and as closed source commercial products. Software platforms are more widely used because of their flexibility as they are easier to deploy them at multiple nodes, have the ability to rapidly modify and extend and the possibility to perform more realistic experiments. Unfortunately, the software implementations are highly unreliable as they are dependent on the commercial off-the-shelf (COTS) hardware used, the operating system adopted, and the status of the host used for traffic generation. Moreover, hardware platforms are typically more precise and reach better performance as they are completely independent of the host computer features and they sometimes run without a computer. The hardware solution that incorporates FPGAs combines high computational capabilities with low energy consumption. FPGA is the easier and lower cost solution of hardware that maintains the performance of the hardware.

This thesis has focused on the implementation of high performance network measuring system based on FPGA using a GPS system for high precision packet time-stamping. A UDP packet generator on NetFPGA 10G platform was successfully implemented that generates dynamically packets without any pre-knowledge and adds the time-stamp at the sending packets using a GPS system that provides high accuracy. In addition, a high precision receptor on the FPGA is used for time-stamping the received packets and the storing them to the hard disk for further offline processing. The ability of our system to generate packets without the pre-capture of any traffic nflows makes it more flexible than other similar implementations.

The measurements revealed that the packet generator can dynamically generate and send time-stamped packets at the theoretical throughput limits of the 802.3 with low jitter. This concludes that the project can measure 10 Gigabit networks with high precision with few packets with the technique of the packet time-stamping. The off-line analysis that provides the projects gives the opportunity to examine the flow more than one time. The limitation of the storing packets that is observed for big and small payload flows is because of the standard hardware module that the NetFPGA hardware library offers. Even this problem can be solved by cutting and soring a small part of big packets. The software implementations have a big impact at the calculated throughput. Even for the simplest process of the packet retransmission, the software reduces the throughput dramatically and adds a high jitter at the packets.

6.1 Future Work

As with almost all areas of research, there exist areas of continual improvement in design and implementation. Here are some aspects for future consideration:

- The current design can produce UDP flows. This can be extended by implementing the TCP protocol on the NetFPGA for dynamic generating of TCP flows. The big size of the FPGA of the NetFPGA 10G permits to be developed a complex protocol like TCP.
- Until now, the calculation of the network performance is done offline. Changes at the receiving part can be performed in order to have live calculation of the network parameters on the hardware. In addition, with this technique it will not have packet losses because of the packet storing.
- The Graphical User Interface (GUI) makes a program more accessible to the end user. A nice friendly GUI can be implemented in order the packet generator to be set easily and to have a nice live visualization of the calculated results.
- Extended testing can be done with more network devices in order to make a ranked performance of more commercial network devices.

Bibliography

[1] C. Liu, "Transmit/receive switch for 10base-t home network," May 21 2002, uS Patent 6,393,050. [Online]. Available: http://www.google.com/patents/US6393050

[2] M. Palmer, *Hands-On Networking Fundamentals*, 2nd ed. Boston, MA, United States: Course Technology Press, 2012, ch. 4.

[3] H. J. Lee, M. S. Kim, J. W. Hong, and G. H. Lee, "QoS Parameters to Network Performance Metrics Mapping for SLA Monitoring," 2002.

[4] A. Botta, A. Dainotti, and A. Pescape, "Do you trust your software-based traffic generator?" *Communications Magazine, IEEE*, vol. 48, no. 9, pp. 158–165, Sept 2010.

[5] "What is an fpga," 2014. [Online]. Available: http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm

[6] "Netfpga," 2014. [Online]. Available: http://netfpga.org/

[7] G. Covington, G. Gibb, J. Lockwood, and N. McKeown, "A packet generator on the netfpga platform," in *Field Programmable Custom Computing Machines, 2009. FCCM '09. 17th IEEE Symposium on*, April 2009, pp. 235–238.

[8] "Ixchariot," 2014. [Online]. Available: http://www.ixiacom.com/products/ixchariot/

[9] "Iperf," 2014. [Online]. Available: http://code.google.com/p/iperf/

[10] "apcom," 2014. [Online]. Available: http://www.apcon.com/products/network-latency-measurement-time-stamping

[11] Z. Zhou, L. Cong, G. Lu, B. Deng, and X. Li, "Hats: High accuracy timestamping system based on netfpga," in *Advances in Computer Science and Information Technology*, ser. Lecture Notes in Computer Science, T.-h. Kim and H. Adeli, Eds. Springer Berlin Heidelberg, 2010, vol. 6059, pp. 183–195. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13577-4_16

[12] M. Shahbaz, G. Antichi, Y. Geng, N. Zilberman, A. Covington, M. Bruyere, N. Feamster, N. McKeown, B. Felderman, M. Blott, A. W. Moore, and P. Owezarski, "Architecture for an open source network tester," *Architectures for Networking and Communications Systems*, vol. 0, pp. 123–124, 2013.

[13] "Osnt," 2014. [Online]. Available: http://osnt.org/

[14] J. J. Garnica, V. Moreno, I. Gonzalez, S. Lopez-Buedo, F. J. Gomez-Arribas, and J. Aracil, "Argos: A gps time-synchronized network interface card based on netfpga," 2010.

[15] "One-way delay measurement techniques," *Accedian Networks*, vol. 1.0, 2010. [Online]. Available: www.accedian.com/en/library/download/1054

[16] N. Krasner, “Method and apparatus for determining time for gps receivers,” Nov. 21 2000, uS Patent 6,150,980. [Online]. Available: http://www.google.com/patents/US6150980

[17] H.-G. Berns, T. Burnett, R. Gran, and R. Wilkes, “Gps time synchronization in school-network cosmic ray detectors,” in *Nuclear Science Symposium Conference Record, 2003 IEEE*, vol. 2, Oct 2003, pp. 789–792 Vol.2.

[18] “spectaracom,” 2014. [Online]. Available: http://www.spectracomcorp.com/

[19] E. A. Hall, *Internet core protocols - the definitive guide: an owner’s manual for the internet.* O’Reilly, 2000.

[20] J. Postel, “Internet Protocol,” RFC 791 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1349, 2474, 6864. [Online]. Available: http://www.ietf.org/rfc/rfc791.txt

[21] R. Braden, “Requirements for Internet Hosts - Communication Layers,” RFC 1122 (INTERNET STANDARD), Internet Engineering Task Force, Oct. 1989, updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633, 6864. [Online]. Available: http://www.ietf.org/rfc/rfc1122.txt

A
Packet Headers

A.1 IEEE 802.3 Ethernet frame

Ethernet is the most common local area networking technology, and, with gigabit and 10 gigabit Ethernet, is also being used for metropolitan-area and wide-area networking. Ethernet refers to the family of LAN products covered by the IEEE 802.3 standard that defines the carrier sense multiple access collision detect (CSMA/CD) protocol. 802.3 specifies the physical media and the working characteristics of Ethernet. Four data rates are currently defined for operation over optical fiber and twisted-pair cables: 10Base-T Ethernet (10 Mb/s), Fast Ethernet (100 Mb/s), Gigabit Ethernet (1000 Mb/s) and 10-Gigabit Ethernet (10 Gb/s).

The IEEE 802.3 standard provides MAC (Layer 2) addressing, duplexing, differential services, and flow control attributes, and various physical (Layer 1) definitions, with media, clocking, and speed attributes.

The figure A.1 shows the complete Ethernet frame, as transmitted, for the payload size up to the Maximum transmission unit (MTU) of 1500 octets. Some implementations of Gigabit Ethernet support larger frames, known as jumbo frames.

- **Preamble:** 64bits. It consists of seven bytes all of the form 10101010. It is used by the receiver to allow it to establish bit synchronization

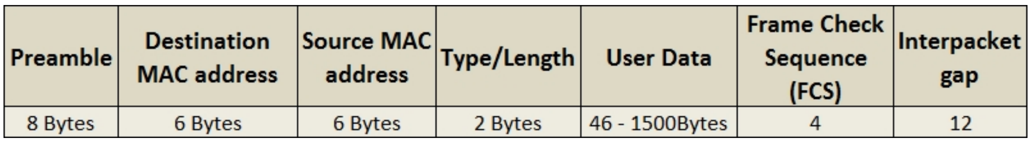
- **Destination MAC address:** 48 bits. This field specifies the receiver MAC address of the packet. A destination MAC address of ff:ff:ff:ff:ff:ff indicates a Broadcast, meaning the packet is sent from one host to any other on that network.

- **Source MAC address:** 48 bits. This field specifies the sender MAC address of the packet.

- **Type / Length field:** 16bits. It can be used for two different purposes. If the type/length field has a value 1500 or lower, it's a length field, otherwise it's a type field and is followed by the data for the upper layer protocol. When the length/type field is used as a length field the length value specified does not include the length of any padding bytes.

- **User Data:** 46 octets - 1500 octets. Non-standard jumbo frames allow for larger maximum payload size.

- **Frame check sequence (FCS):** 32bits. It is a 4-octet cyclic redundancy check which allows detection of corrupted data within the entire frame.
- **Interpacket gap:** 96bits. Idle time between packets. After a packet has been sent, transmitters are required to transmit a minimum of 96 bits (12 octets) of idle line state before transmitting the next packet.



Preamble	Destination MAC address	Source MAC address	Type/Length	User Data	Frame Check Sequence (FCS)	Interpacket gap
8 Bytes	6 Bytes	6 Bytes	2 Bytes	46 - 1500Bytes	4	12

Figure A.1: 802.3 Ethernet packet

A.2 IPv4 Network Layer

IPv4 is a protocol that is used for routing the data packets between different auto-organized networks. It operates on a best effort delivery model, in that it does not guarantee delivery, nor does it assure proper sequencing or avoidance of duplicate delivery. The internet protocol uses five key fields/values in the header in providing its service:IP address, Type of Service, Time to Live, and Header Checksum.

Each device that participate in a computer network should have an IP address. With this way the hosts and the network interfaces are identified. An address indicates where the device is.

The Type of Service mechanism is used to indicate the quality of the service desired by changing the priority of the packet queuing and routing on the routers.

Time to Live is an indication of an upper bound on the lifetime of an internet datagram. It is set by the sender of the datagram and reduced at the points along the route where it is processed. If the time to live reaches zero before the internet datagram reaches its destination, the internet datagram is destroyed. The time to live mechanism can be thought of as a self destruct time limit.

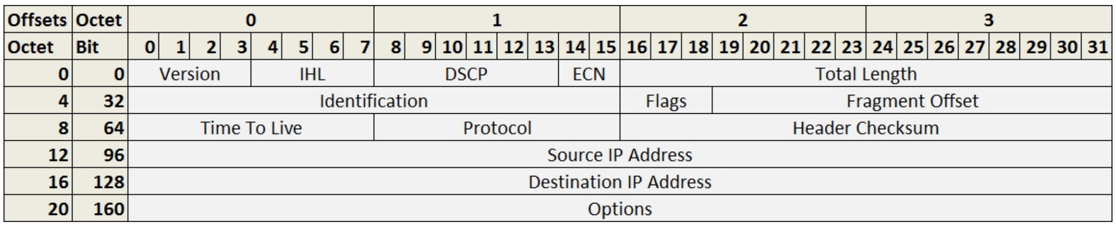
The Header Checksum provides a verification that the information used in processing internet datagram has been transmitted correctly. The data may contain errors. If the header checksum fails, the internet datagram is discarded at once by the entity which detects the error.

The internet protocol does not provide a reliable communication functionality. There are no acknowledgements either end-to-end or hop-by-hop. There is no error control for data, no retransmissions and no flow control. [19]

The header of the IPv4 is consist of 20 bytes and 13 fields as it is shown at the figure A.2. [20]

- **Version:** 4 bits. Indicates the version of the IP. Always assigned to 4.
- **Internet Header Length (IHL):** 4 bits. Size of IP header in 32-bit words. The minimum value is 5, which is a length of 5x32 = 160 bits = 20 bytes. The maximum value is 15 words, 15x32 =480 bits = 60 bytes.
- **Differentiated Services Code Point (DSCP):** 6 bits. Originally defined as the Type of service field.
- **Explicit Congestion Notification (ECN):** 2 bits. ECN is an optional feature that allows end-to-end notification of network congestion without dropping packets.

- **Total Length:** 16 bits.It defines the entire packet (fragment) size, including header and data, in bytes. The minimum-length packet is 20 bytes (20-byte header + 0 bytes data) and the maximum is 65,535 bytes. The largest datagram that any host is required to be able to reassemble is 576 bytes, but most modern hosts handle much larger packets.
- **Identification:** 16 bits. An identifying value assigned by the sender to aid in assembling the fragments of a datagram.
- **Various Control Flags:** 3 bits. Bit 0: reserved, must be zero Bit 1: 0 = May Fragment, 1 = Don't Fragment. Bit 2: 0 = Last Fragment, 1 = More Fragments.
- **Fragment Offset:** 13 bits. This field indicates where in the datagram this fragment belongs. The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.
- **Time to Live:** 8 bits. This field indicates the maximum time the datagram is allowed to remain in the internet system. If this field contains the value zero, then the datagram must be destroyed. This field is modified in internet header processing.The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.
- **Protocol:** 8 bits. This field indicates the next level protocol used in the data portion of the internet datagram.
- **Header Checksum:** 16 bits. A checksum on the header only. Since some header fields change (e.g., time to live), this is recomputed and verified at each point that the internet header is processed.
- **Source Address:** 32 bits. This field is the IPv4 address of the sender of the packet.
- **Destination Address:** 32 bits. This field is the IPv4 address of the receiver of the packet.



Offsets	Octet	0								1								2								3							
Octet	**Bit**	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset												
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options																															

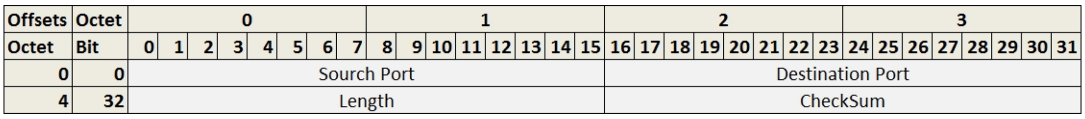
Figure A.2: IP header's fields

A.3 UDP Transport Layer

User Datagram Protocol(UDP) together with Transmission Control Protocol(TCP) are the two types of Internet Protocol. UDP is not a connection oriented protocol as TCP. Data can be sent bidirectionally with no more effort. Multiple messages are sent as packets in chunks using UDP. UDP protocol is used in DNS, TFTP, SNMP, RIP, VOIP packets and it is very simple because it does not have an inherent order as all of its packets are independent from each other. Moreover, it does not use acknowledgements to check the missing packets; it does not have flow control and, lastly, it does not need handshake for establishing the connection. If ordering or reliability is required, it has to be managed by the application layer. [21]

The header of UDP is 8 byte length and consists of 4 fields of 16 bits, Source port, Destination port, length and checksum as it shows the figure A.3.

- **Source Port:** 16bits. It indicates the port of the sending process, and may be assumed to be the port to which a reply should be addressed in the absence of any other information. If not used, a value of zero is inserted.
- **Destination Port:** 16bits. This field identifies the receiver's port and is required. Similar to source port number, if the client is the destination host then the port number will likely be an ephemeral port number and if the destination host is the server then the port number will likely be a well-known port number.
- **Length:** 16bits. Length is the length in octets of this user datagram including this header and the data.The minimum value of the length is eight.
- **Checksum:** 16bits. The checksum field is used for error-checking of the header and data. It is computed from the IP header, UDP header and the data. If the checksum is cleared to zero, then check summing is disabled. If the computed checksum is zero, then this field must be set to 0xFFFF.



Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Sourch Port																Destination Port															
4	32	Length																CheckSum															

Figure A.3: UDP header's fields

B Xilinx's Protocol Handshakes

B.1 AXI4-STEAM

The AXI4-Stream Interconnect is a key Interconnect Infrastructure IP which enables connection of heterogeneous master/slave AMBA AXI4-Stream protocol compliant endpoint IP. The AXI4-Stream Interconnect routes connection from one or more AXI4-Stream master channels to one or more AXI4-Stream slave channels. The protocol gives the ability for multiple master to multiple slave (up to 16x16) configuration implementing a cross-point switch. In addition the clock rate conversion can be synchronous and asynchronous as multiple clock domains are supported.

Figure B.1 shows the transfer of data in an AXI4 sream channel. TVALID is driven by the source side of the channel and TREADY is driven by the receiver . TVALID indicates that the value in the payload fields (TDATA, TUSER and TLAST) is valid. TREADY indicated that the slave is ready to receive data. When bot TVALID and TREADY are true in a cycle, a transfer occurs. The master and slave will set TVALID and TREAD respectively for the next transfer appropriately.

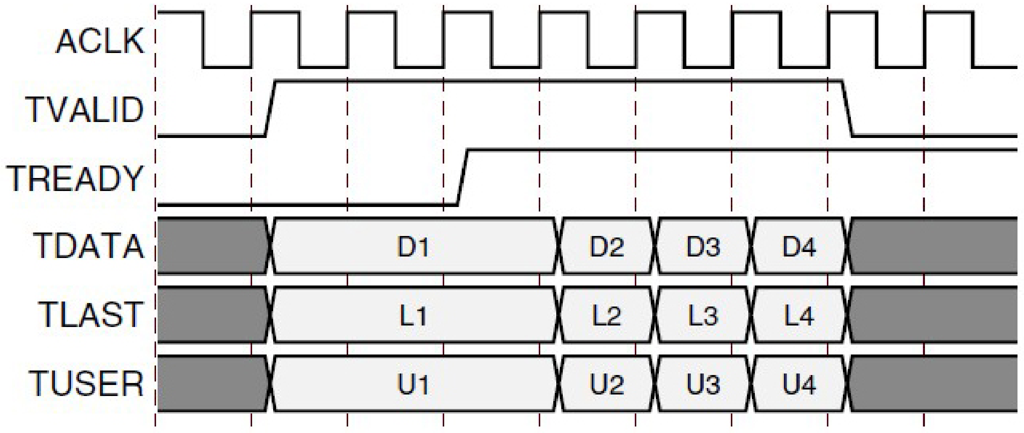


Figure B.1: Data Transfer in an AXI-Stream Channel