DESIGN AND IMPLEMENTATION OF SCHEDULING AND SWITCHING ARCHITECTURES FOR HIGH SPEED NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUSTAFA SANLI

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

OCTOBER 2011

Approval of the thesis:

DESIGN AND IMPLEMENTATION OF SCHEDULING AND SWITCHING ARCHITECTURES FOR HIGH SPEED NETWORKS

submitted by **MUSTAFA SANLI** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _________

Prof. Dr. İsmet Erkmen
Head of Department, **Electrical and Electronics Engineering** _________

Prof. Dr. Hasan Cengiz Güran
Supervisor, **Electrical and Electronics Engineering Dept., METU** _________

Assoc. Prof. Dr. Ece Güran Schmidt
Co-Supervisor, **Electrical and Electronics Engineering Dept., METU** _________

Examining Committee Members:

Prof. Dr. Semih Bilgen
Electrical and Electronics Engineering Dept., METU ___________________

Prof. Dr. Hasan Cengiz Güran
Electrical and Electronics Engineering Dept., METU ___________________

Assoc. Prof. Dr. Cüneyt Bazlamaçcı
Electrical and Electronics Engineering Dept., METU ___________________

Assoc. Prof. Dr. Nail Akar
Electrical and Electronics Engineering Dept., Bilkent Univ. ___________________

Assist. Prof. Dr. İlkay Ulusoy
Electrical and Electronics Engineering Dept., METU ___________________

Date: October 6, 2011

ABSTRACT

DESIGN AND IMPLEMENTATION OF SCHEDULING AND SWITCHING ARCHITECTURES FOR HIGH SPEED NETWORKS

Sanlı, Mustafa
Ph. D., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Hasan Cengiz Güran
Co-Supervisor: Assoc. Prof. Dr. Ece Güran Schmidt

October 2011, 122 pages

Quality of Service (QoS) schedulers are one of the most important components for the end-to-end QoS support in the Internet. The focus of this thesis is the hardware design and implementation of the QoS schedulers, that is scalable for high line speeds and large number of traffic flows. FPGA is the selected hardware platform.

Previous work on the hardware design and implementation of QoS schedulers are mostly algorithm specific. In this thesis, a general architecture for the design of the class of Packet Fair Queuing (PFQ) schedulers is proposed. Worst Case Fair Weighted Fair Queuing Plus (WF^{2}Q+) scheduler is implemented and tested in hardware to demonstrate the proposed architecture and design enhancements.

The maximum line speed that PFQ algorithms can operate decreases as the number of scheduled flows increases. For this reason, this thesis proposes to aggregate the flows to scale the PFQ architecture to high line speeds. The Window Based Fair Aggregator (WBFA) algorithm that this thesis suggests for flow aggregation provides a tunable trade-off between the efficient use of the available bandwidth and the fairness among the constituent flows. WBFA is also integrated to the hardware PFQ architecture.

The QoS support provided by the proposed PFQ architecture and WBFA is measured by conducting hardware experiments on a custom built high speed network testbed which consists of three data processing cards and a backplane. In these experiments, the input traffic is provided by the hardware traffic generator which is designed in the scope of this thesis.

Keywords: Quality of Service Scheduler, High speed network, Flow aggregation

ÖZ

YÜKSEK HIZLI AĞLAR İÇİN ZAMANLAMA VE ANAHTARLAMA MİMARİLERİNİN TASARIMI VE GERÇEKLENMESİ

Sanlı, Mustafa
Doktora, Elektrik Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Hasan Cengiz Güran
Ortak Tez Yöneticisi: Doç. Dr. Ece Güran Schmidt

Ekim 2011, 122 sayfa

Servis kalitesi (QoS) çizelgeleyiciler, internette uçtan uca QoS desteği için en önemli bileşenlerdendir. Bu tezin odaklandığı konu, yüksek hat hızlarına ve çok sayıda trafik akışına ölçeklenebilen QoS çizelgeleyicilerin donanımsal tasarımı ve gerçeklenmesidir. Seçilen donanım platformu FPGA'dir.

QoS çizelgeleyicilerin donanımsal tasarımı ve gerçeklenmesi üzerine yapılan önceki çalışmalar çoğunlukla algoritmaya özeldir. Bu tezde Paket Adil Kuyruklama (PFQ) sınıfındaki çizelgeleyicilerin tasarımı için genel bir mimari önerilmiştir. Bu sınıftaki çizelgeleyicilerden bir tanesi, önerilen mimariyi ve tasarım iyileştirmelerini örnek üzerinde göstermek için donanım üzerinde gerçeklenmiş ve test edilmiştir.

Çizelgelenen akış sayısı arttıkça PFQ algoritmalarının çalışabildiği en yüksek hat hızı azalmaktadır. Bu yüzden, bu tez PFQ mimarisini yüksek hat hızlarına ölçeklendirmek için akışları birleştirmeyi önermektedir. Bu tezde akış birleştirme için önerilen Pencere Tabanlı Adil Birleştirici (WBFA) algoritması, mevcut bant genişliğinin etkin kullanımı ile bileşen akışlar arasındaki adillik arasında ayarlanabilir bir ödünleşim sunmaktadır. WBFA aynı zamanda donanımsal PFQ mimarisine de entegre edilmiştir.

Önerilen PFQ mimarisi ve WBFA tarafından sağlanan QoS desteği üç veri işleme kartı ve bir anakarttan oluşan özel üretilmiş bir yüksek hızlı ağ test ortamında donanımsal deneyler yapılarak ölçülmüştür. Bu deneylerde, giriş trafiği bu tez kapsamında tasarlanan bir donanımsal trafik üretici tarafından oluşturulmuştur.

Anahtar Kelimeler: Servis kalitesi çizelgeleyici, Yüksek hızlı ağ, Akış birleştirme

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis advisor Prof. Dr. Hasan Cengiz Güran and my co-advisor Assoc. Prof. Dr. Ece Güran Schmidt for their invaluable support, guidance, encouragement and friendship throughout the completion of the thesis. I would like to thank Prof. Dr. Semih Bilgen, Assoc. Prof. Dr. Cüneyt Bazlamaçcı, Assoc. Prof. Dr. Nail Akar and Assist. Prof. Dr. İlkay Ulusoy for their precious contributions to my thesis. I must acknowledge Aselsan AŞ for providing me time and facilities for my research. Finally, I would like to thank my family for their continuous support throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS	
ASIC	: Application Specific Integrated Circuit
ATM	: Asynchronous Transfer Mode
BSFQ	: Bin Sort Fair Queuing
BWRR	: Budgeted Weighted Round Robin
CAI	: Work-Conserving Aggregation with Isolation
CDF	: Cummulative Distribution Function
CF	: Continuous Framing
CLB	: Configurable Logic Block
DDR	: Double Data Rate
D-EDD	: Delay Earliest Due Date
DRAM	: Dynamic Random Access Memory
FAbS	: Flow Aggregate based Services
FAN	: Flow Aware Networking
FCFS	: First Come First Served
FIFO	: First In First Out
FPGA	: Field Programmable Gate Array
FPGEN	: Fast Packet GENerator
FQ	: Flag Queue

FRR	: Fair Round Robin
FSA	: Flow State Aware Architecture
GFS	: Gigabit Fair Scheduling
GPS	: Global Processor Sharing
GR	: Guaranteed Rate
HOL	: Head Of Line
HRR	: Hierarchical Round Robin
IP	: Internet Protocol
KS	: Kolmogorov-Simirnov
LFSR	: Linear Feedback Shift Register
LR	: Latency Rate
MMBP	: Markov Modulated Bernoulli Process
NGN	: Next Generation Network
PCAP	: Packet CAPture
PCB	: Printed Circuit Board
PCI	: Peripherial Component Interconnect
PFQ	: Packet Fair Queuing
PPS	: Packets Per Second
PTG	: Precise Traffic Generator
QoS	: Quality of Service
RAM	: Random Access Memory
RCSP	: Rate Controlled Static Priority
RED	: Random Early Detection
RSE	: RAM-based Searching Engine

S&G	: Stop and Go
SCFQ	: Self Clocked Fair Queuing
SFQ	: Start time Fair Queuing
SRAM	: Static Random Access Memory
SRR	: Stratified Round Robin
TCP	: Transmission Control Protocol
TCRM	: Traffic Controlled Rate Monotonic priority
TSFQ	: Tiered Service Fair Queuing
VHDL	: Very high speed integrated circuit Hardware Decription Language
VOIP	: Voice Over IP
WBFA	: Window Based Fair Aggregator
WF^2Q	: Worst Case Fair Weighted Fair Queuing
WF^2Q+	: Worst Case Fair Weighted Fair Queuing Plus
WFI	: Worst case Fairness Index
WFQ	: Weighted Fair Queuing
XOR	: Exclusive OR

CHAPTER 1

INTRODUCTION

The services in the Next Generation Network (NGN) include real time applications such as IP telephony and video in addition to virtual environments and global or local information centers. These services will be created on demand by the customers and will be carried out with end-to-end Quality of Service (QoS) support such as bandwidth, delay and jitter guarantees [1]. The end-to-end QoS requires the classification of packets into flows on each network node on their path, storing them in per flow queues and employing a scheduler to decide for the service received by these queues. These are all data path operations which have to be executed for each packet hence they are implemented in hardware and their complexity affects the feasibility of the implementation.

The scheduling algorithms which are used in the QoS schedulers of the routers and the switches are one of the most important components determining the Quality of Service (QoS) performance of the Internet. In order to provide QoS support, the scheduling algorithms specify the order of transmission for the the packets that are queued at the output ports. The scheduling algorithms enable different services for different flows and affect the overall QoS closely. A good scheduling algorithm should efficiently utilize the network resources and provide protection between the flows. This protection should prevent some greedy flows from taking the service share of the other flows. The scheduling algorithm should also provide fairness by serving the flows in proportion to the agreed service shares. Another property that should be found in a good scheduling algorithm is flexibility. This property allows the algorithm to support different applications. While possessing these properties, the scheduling algorithm should be simple at the same time. This simplicity provides low algorithmic complexity and also low implementation complexity.

Generalized Processor Sharing (GPS) is an ideal hypothetical scheduling algorithm which can provide perfect protection among the flows [2, 3]. GPS is based on a fluid flow model where traffic flows are infinitely divisible and multiple flows can receive service simultaneously and the service share of each flow is proportional to its weight. GPS can provide network delay bound for leaky bucket constrained traffic [4-6]. However, in packet networks, the packet is the minimum service unit. As a result of this, GPS cannot be realized in packet networks.

With the intent of getting as close as possible to this ideal scheduling, a class of Packet Fair Queuing (PFQ) algorithms is proposed to emulate the behavior of GPS [7-17]. In all PFQ algorithms, there is a global function called virtual time which is used to track the progress of the GPS scheduler. For each head of line (HOL) packet of each flow in the system, a finish time is calculated. This finish time corresponds to the time that this packet would leave the GPS scheduler. Packets are then served in the order of their respective finish times.

All the PFQ algorithms, even the simpler ones have implementation difficulties which put a limit on the maximum number of flows supported for a fixed amount of implementation area. Calculations of the timestamp functions require hardware division. Also the scheduler needs to search the minimum time stamp value among many flows. The logic resources that is necessary to accomplish these tasks increase with the increasing number of flows.

In this thesis, different than the previous works that present designs for specific PFQ algorithms we propose a general framework that can be used to implement any given PFQ algorithm. To this end, we provide a block level architecture which separates the general components that are common to all PFQ schedulers and the algorithm-specific components. In this architecture, we identify the design challenges and use techniques such as look-up-table based operations and dynamical adaptation of these tables to overcome these difficulties. We use our proposed architecture to implement a popular PFQ algorithm, i.e., Worst Case Fair Weighted Fair Queing Plus (WF^{2}Q+) [16] on hardware. The algorithm is implemented on an FPGA (Field Programmable Gate Array) based board and the performance evaluation is performed on a hardware testbed. The main reason for selecting the FPGA as the hardware implementation medium is that the very high number of logic gates and embedded blocks in today's FPGAs enables the design of complex hardware platforms with reduced engineering cost and rapid turnaround time.

The complexity of the scheduling algorithms increases with the quality of the provided service. Furthermore the complexity increases with the number of flows that are scheduled. The schedulers are required to operate at wire speed and the execution of the scheduling algorithm has to be completed in a single packet time. As a consequence, when the number of flows exceeds a certain limit, the scheduling algorithm will be unable to continue its proper operation.

The commonly proposed solution for increasing the number of flows supported by the scheduler is simply employing the latest hardware technology to achieve the fastest implementation. However, the number of flows is always increasing with the new types of applications and the increasing number of devices connected to the Internet. Hence there will always be a limit on the number of flows supported by a given architecture. In addition, the cost of implementing such high-speed, high-capacity router with cutting edge technology is very high. Considering the dynamic network traffic profile with changing loads and flow patterns, it is expected that most of the time the router will be operating with smaller number of flows than it can actually support. This will lead to inefficient use of the investment.

Another approach for building packet schedulers that support large number of flows is employing pipeline techniques which require less hardware resources in the expense of increased time to process each packet. In these approaches, the packets arriving at the instances of timestamp calculation and packet selection are discarded. In high data rates, the number of discarded packets can reach to an order of ten thousands [18]. Decreasing the per packet processing time in the pipelined approach requires expensive hardware resources as discussed in [19] which again leads to high cost implementations.

A third approach to support high number of flows is aggregating them to decrease the implementation complexity. The basic problem in flow aggregation is preserving QoS guarantees of the constituent flows in the aggregate. As a result of the greedy behavior of one of these flows, the others may receive decreased delay and fairness performance. This problem is tackled in [20] which proposes a network model that consists of flow aggregators and packet schedulers. In this work, it is proved that if the flow aggregation is performed fairly and the packet schedulers have certain properties, the end-to-end delay guarantees are preserved with respect to the case that no flow aggregation is performed. [20] presents two different approaches for the design of fair aggregators. The first one is "the basic fair aggregator" which limits the service rate for the aggregated flow to the sum of the reserved rates of the input flows. The second approach is "the greedy fair aggregator" which relaxes this limit only if all input flows have an arrival rate greater than their reserved rates. It is possible that the arrival rate of the flows to be aggregated exceed the total reserved rate temporarily. In such case even if there is available capacity to serve these flows, it will not be utilized.

In this thesis, we present Window Based Fair Aggregator (WBFA) and analytically show that it is a fair aggregator as defined in [20]. Hence, the individual delay bounds of the constituent flows aggregated by WBFA are preserved. Our approach allows the constituent flows to use the full capacity of the output channel until the difference in the service received by the flows reaches a limit. As a result of the increase in the utilization, the average end-to-end delays provided by WBFA are expected to be lower than the basic and greedy fair aggregators proposed by [20]. While increasing the utilization, WBFA also preserves the fairness of service to the aggregated flows. WBFA provides a tunable trade-off between the efficient use of the available bandwidth and the fairness among the constituent flows. In addition to these, WBFA has low implementation complexity and can be efficiently implemented on hardware. WBFA is implemented on an FPGA based board and its performance is measured in our hardware testbed.

The hardware testbed is designed in the context of this thesis. The testbed consists of several FPGA based boards and a backplane to connect these boards. The schematic design of the boards, PCB layout design, and manufacturing of the boards are done as a part of this thesis.

A traffic generator “FPGEN (Fast Packet GENerator)” [21] is designed to measure the performance of the schedulers that are implemented on the hardware testbed. FPGEN is a programmable random traffic generator which is entirely implemented on FPGA. FPGEN can generate variable packet size Internet traffic with Poisson and Markovian arrivals at OC-48 rate per interface. We present a model which overcomes the inherent difficulties of generating Poisson traffic on a serial interface due to the required independency between the packet sizes and the inter-packet times. In addition, FPGEN can generate Markov-modulated traffic entirely on hardware.

FPGEN is scalable to high-speeds as it is implemented purely on hardware without using any high level programming or processors. The packet generation times are randomly computed in real-time entirely using the logic resources of the FPGA. The FPGEN board has two OC-48 fiber-optical interfaces and operates at 125MHz. Hence, it is able to support a total traffic generation rate close to 5 Gbps and 250 million packets per second. FPGEN is configurable to generate traffic with different parameters due to the programmability of the FPGA. Our research on FPGEN includes the theoretical design of FPGEN, the hardware design of the FPGA-based traffic generator board and the implementation of FPGEN on FPGA.

The novel contributions of this thesis can be listed as follows.

A traffic generator “FPGEN” which can generate Poisson traffic and Markov-modulated on-off traffic is designed. The traffic generator is scalable to high speeds as a result of the operations being purely carried on hardware [21].

Generating traffic according to given statistics on a serial interface has inherent difficulties due to the required independence between the packet sizes and the inter-packet times. We present a model which can overcome these difficulties for Poisson traffic and be implemented on hardware. To the best of our knowledge there is no other published work on a hardware-based packet generator that produces Poisson traffic with exponentially distributed packet sizes.

The previous approaches for generating Markov-modulated traffic include RAM based and processor based techniques. We apply our design approach to generate on-off traffic entirely on hardware.

For FPGEN, we provide hardware design details and performance measurement results that demonstrate the achieved rate and the statistical properties of the generated traffic.

We propose a general hardware architecture for the design of the family of PFQ schedulers. We use this architecture to identify the design challenges. We propose new design improvements and use previously presented approaches to overcome these difficulties.

Using our proposed architecture, we implement the WF^{2}Q+ algorithm on hardware. We provide hardware design details. We make performance measurements and provide test results. We show that the results are within the theoretical limits.

We propose a novel flow aggregation algorithm "WBFA" and show analytically that WBFA is a fair aggregator. We calculate theoretical delay bounds for WBFA.

We implement WBFA on hardware and make performance measurements. We present the test results and show that the results agree with the theoretical delay bound.

The thesis is organized in seven chapters. In Chapter 2, QoS schedulers are described by introducing the basic concepts used throughout the thesis. Chapter 3 gives the details of the design of the hardware testbed. In Chapter 4, FPGEN traffic generator is explained in detail. Chapter 5 explains the design of the proposed PFQ architecture. Both theoretical design details and implementation results are given in this chapter. In Chapter 6, Window Based Fair Aggregator design is introduced. In this chapter, first it is shown analytically that WBFA is a fair aggregator. Then the hardware implementation of the WBFA is explained. Finally Chapter 7 summarizes the thesis and presents the conclusions.

CHAPTER 2

SWITCHES FOR HIGH SPEED NETWORKS

Everyday, Internet offers new benefits for the welfare of mankind. We have already adopted ourselves to file sharing, voice conversation, video meeting and streaming audio and video applications over the internet. The widespread usage of Voice Over IP (VOIP) and high definition video broadcasting is also on the way. When all of these applications are summed up, we need quite huge network bandwidth and strict quality of service support for the sake of being more tightly connected.

To support the demand for network bandwidth, optical fiber technology has grown very fast and found large application areas especially in backbone networks. Optical fiber technology enables the transmission of multi-gigabit data in one second over a single fiber line. By using many fiber lines in parallel and also adopting optical technologies such as wavelength converters, very high data rates are offered to support the growing bandwidth requirement. Due to the recent developments in optical fiber technology, huge carrying capacities are provided to the computer networks.

The widespread usage of the Internet and computer services has resulted in a rapid increase in the network traffic. In this traffic, an important part of the services require real-time data transmission. Audio and video streaming, video conferencing, internet telephony are a few examples of these kind of multimedia services. These services require large amount of network bandwidth and QoS support such as delay and jitter bounds and throughput guarantees.

In order to provide QoS support for the real-time services and also to utilize the resources such as bandwidth and buffers efficiently, packets should be assigned different priorities when accessing network resources. End to end QoS can be provided by allocating the network resources among flows according to the type of the packet data or the type of service that is purchased by the customer. In order to achieve this, the packets arriving from different flows are kept in separate queues. The QoS scheduler makes a choice among the head of line packets in these queues and selects the packet that is to be transmitted next.

QoS scheduling is a data plane function and is performed on every single packet in a computer network [22-24]. The data plane functions are required to be executed without slowing down the data transmission in a network device. This wire speed operation of a network device demands for extremely short packet processing times. In order to reach high data rates, packet processing applications such as classification, table look-up and header modification should take place in embedded hardware platforms. Packet buffering and buffer management requirements had further increased the importance of efficient implementation of schedulers in hardware.

2.1 SWITCHES AND ROUTERS

Internet is in fact, network of networks which are composed of millions of computing devices connected with communication links such as copper, radio, fiber, satellite, etc... In these networks, the data is routed among the end systems by the help of routers. The main job of a router is forwarding the packets from source to the recipients. Figure 2-1 shows a typical interconnected network with several end systems and routers.

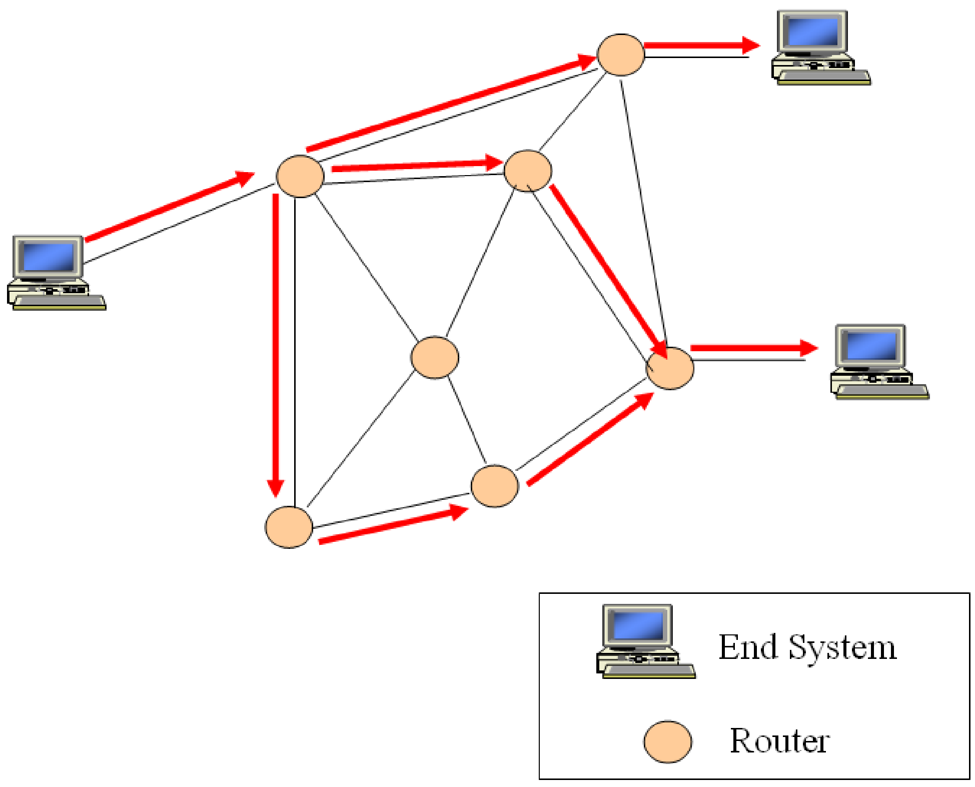


Figure 2-1 Typical interconnected network with several end systems and routers.

When a packet is received, the router first looks up the packet destination address in the forwarding table to identify the outgoing ports. Then, it manipulates the header according to the needs and sends the packet to the output ports. In the output ports, the packet is queued and finally transmitted onto the outgoing link.

While a router is a layer-3 device, a switch is a layer-2 device that operates on Ethernet frames. The hardware of both devices is similar but the router has additional layer-3 software. Hence, in this thesis, our focus will be on the network switches.

Generic switch architecture is composed of input and output ports which are called line cards, switch fabric and CPU. Figure 2-2 shows the structure of a generic router. Line cards are entry and exit points of data in the router. They connect external network to switch fabric. Many physical layer actions such as signal conversions between different domains, synchronization and frame processing take place in the line cards. Switch fabric connects the input and output ports of the router and performs the task of switching. After switching, the packets are queued at the buffers of the output line cards. The packet that will be delivered to the output line is selected by the QoS scheduler at the output line card. The goal of the QoS scheduler is to provide different service and different priorities for different traffic sources. CPU deals with general management and maintenance of the router such as updating address tables, collecting packet statistics, etc...

One of the most important characteristics of the router architecture is the switch fabric's speedup. Speedup is defined as the ratio of the data rate that can be switched to an output port to line rate. If the speedup is 1, that is the switch fabric speed is the same as that of the network line, all the packets are queued at the input line cards waiting for the switching fabric. Assuming that the router has *N* input and *N* output ports, if the speedup is greater than 1 but smaller than *N*, some of the packets will wait for the switch fabric in the buffers at the input line cards and some of them will wait for the QoS scheduler in the buffers at output line cards. Ideal performance is achieved when speedup is *N*. In this case, packets do not need to wait for the switch fabric because the switch fabric is fast enough to serve the packets coming from all the ports as if there were only one input port. The packets are queued only in the buffers at the output line cards of the router. When speedup is *N*, the QoS scheduler has access to all the packets waiting in the router, hence "speedup=*N*" is the best case for the QoS support.

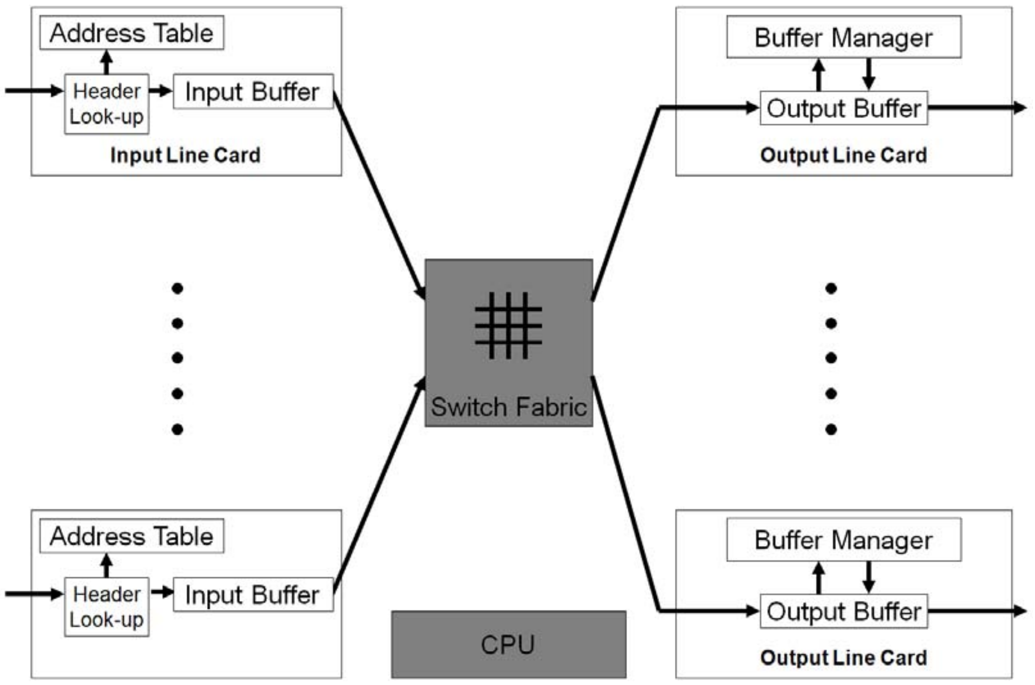


Figure 2-2 The structure of a generic router.

2.2 QUALITY OF SERVICE SCHEDULERS

Quality of service is the ability to have resource guarantees and service differentiation so that the applications which are delay, jitter or loss sensitive can perform satisfactorily. QoS can be provided by giving relative priorities and defining different levels of service to different flows and packets in the network. In order to provide QoS support, the QoS scheduler specifies the order in which the packets queued at the output ports are actually transmitted. The QoS scheduler gives different service to different connections.

In today's switches, increasing QoS requirements have put a significant emphasis on the design of schedulers. Schedulers are generally evaluated using performance metrics such as complexity, delay bound, worst case fairness index (WFI) and required buffer space. The complexity is related with the computational resources required for the execution of the scheduling algorithm. Delay bound is the highest possible delay that a packet can encounter under defined traffic conditions. WFI is a parameter that is used to measure the discrepancy between the scheduling algorithm and GPS. Required buffer space tells the amount of packet data that needs to be buffered in the scheduler.

QoS schedulers have two main classes. These are sorted priority based and frame based schedulers. The sorted priority based scheduler computes a timestamp for each arriving packet with respect to current system state and the system is updated accordingly. The scheduler sorts the packets based on their timestamps. This type of schedulers provides tight end-to-end delay bounds. However, computation of the timestamp for each packet, maintaining priority queues and performing computations at line rate results in high complexity. Weighted Fair Queuing (WFQ) [2, 3], Self Clocked Fair Queuing (SCFQ) [7], Delay Earliest Due Date (D-EDD) [25], Rate Controlled Static Priority (RCSP) [26] and Traffic Controlled Rate Monotonic Priority Scheduling (TCRM) [27] are examples of sorted priority based schedulers. All PFQ schedulers are sorted priority based schedulers.

The frame based scheduler splits time into frames and limits the amount of traffic that can be transmitted during a frame period [10]. There might be an additional delay component to smooth the bursts over the frames. This type of schedulers can provide bandwidth guarantees and have low complexity. Stop and Go (S&G) [28], Hierarchical Round Robin (HRR) [29], Continuous Framing (CF) [30] and Budgeted Weighted Round Robin (BWRR) [31] are examples of frame based schedulers.

To ensure the QoS requirements, the traffic has to be shaped and defined according to special traffic models before entering the network. (r, T), (σ, ρ), and $(X_{min}, X_{ave}, I, S_{max})$ are widely used traffic models. In (r, T) model, r is a measure of the average data rate. In an interval of length T, no more than rT bits are transmitted. In (σ, ρ)

model, σ indicates the maximum burst size and ρ indicates the long term bounding rate. In an interval of length T, no more than ($\sigma + \rho T$) packets are transmitted. In (X_{min}, X_{ave}, I, S_{max}) model, X_{min} denotes the minimum inter-arrival time between the packets and X_{ave} denotes the average inter-arrival time between the packets measured over an interval of length I. S_{max} denotes the minimum packet size. In some cases, there may be a need for shaping the traffic at each scheduler. To achieve this, each scheduler has a traffic regulator.

2.2.1 Packet Fair Queuing Schedulers

The ideal scheduling is provided by the hypothetical GPS scheduler. GPS uses a fluid flow model and assumes that the flows are infinitely divisible and multiple flows can receive service at the same time. Despite its ideal scheduling performance, GPS cannot be used in packet switching networks where packet is the smallest service unit. There are different schedulers introduced for use in packet networks. PFQ schedulers try to emulate the behavior of GPS in packet networks.

PFQ schedulers work with (σ, ρ) traffic model. The scheduler first computes the time the packet would complete service when all the connections receive fair service. This value is called "finish number". All the packets are served with this order. The scheduling algorithm is priority based. The finish numbers need to be ordered. The provided end to end delay bound increases with the number of switches on the route.

Weighted Fair Queuing (WFQ) [2, 3] algorithm is known to be the first PFQ algorithm. As long as a flow is leaky bucket constrained, WFQ can provide end-to-end delay bound similar to the GPS. It is proven that WFQ does not fall behind the GPS by more than one maximum size packet but it can be ahead of the GPS [15]. The complexity of WFQ is O(V) where V denotes the number of available connections. The complexity comes from computing the timestamp and maintaining the priority queues. The buffer requirement increases in each switch on the route. Due to the high complexity of the algorithm, implementation is very difficult.

Worst Case Fair Weighted Fair Queuing (WF^{2}Q) algorithm [15] tries to improve WFQ by using an eligibility test in the selection of the packets. When the scheduler is selecting a packet for transmission, the scheduler considers only the eligible packets which are the set of packets that have started service in the emulated GPS system. WF^{2}Q can provide almost the same service as the GPS. The maximum service difference is one maximum packet size [15].

Both WFQ and WF^{2}Q have a major drawback of computational complexity. The time complexity of both of the algorithms is O(N) where N is the number of flows [23]. WF^{2}Q+ [16] is an enhanced version of WF^{2}Q and it has less time complexity. WF^{2}Q+ computes the virtual time function without emulating the GPS. As a result of this, it can provide worst case fairness properties with utilizing simpler calculations. This leads to increased implementation efficiency on hardware platforms.

Several other scheduling algorithms are also proposed to emulate GPS behavior in different ways. SCFQ [7] and Start Time Fair Queuing (SFQ) [32] try to simplify the emulation of GPS by using efficient virtual time functions. Bin Sort Fair Queuing (BSFQ) [33], Stratified Round Robin (SRR) [34], Fair Round Robin (FRR) [35] and Tiered Service Fair Queuing (TSFQ) [36] uses quantization to simplify the emulation.

CHAPTER 3

HARDWARE TESTBED

In this thesis, QoS schedulers are designed and implemented in hardware. Also, new architectures are designed and their performance is evaluated on FPGA. In order to use in our hardware implementations and performance measurements, a digital hardware testbed is designed and produced.

3.1 IDENTIFICATION OF THE DESIGN REQUIREMENTS

The network testbed will contain several data-processing boards and a backplane to organize the communication between the boards. Each one of the data-processing boards should contain a processor for the generation, processing and scheduling of packets, a flash memory to keep non-volatile data, rs-232 interface to connect the board to a PC, DC power converters to produce the different voltage levels required on the board, VME-64 connectors for backplane connection and a PCB designed with special care to prevent signal coupling between the lines at high data rates. Also, fiberoptical transceivers should be used to reach high data rates.

3.1.1 Selection of the Processor

In the implementation of network applications, the choice of the hardware platform affects speed, cost, design time, area requirement, power dissipation and maintenance capabilities. Network applications are implemented on several device families such as general purpose processors, embedded reduced instruction set computer (RISC) processors, network processors, application specific integrated circuits (ASIC), and field programmable gate arrays (FPGA). Each of these families of devices offers specific benefits and drawbacks. A comparison of these families is provided below.

3.1.1.1 General Purpose Processors

General purpose processors are preferred because they are very well-known, they are rather cheap and they offer very elastic usage. However, these processors are not optimized for the network operations. The implementations on these processors cannot take the advantage of bit-level parallelism and concurrency. Their memory access time is rather long. As a result of this, they cannot succeed at high line speeds.

3.1.1.2 Embedded RISC Processors

They are preferred because of their low power dissipation and small area requirement. Nevertheless, they have the same drawbacks as that of general purpose processors. Also, their operating frequency is lower than general purpose processors [37].

3.1.1.3 Network Processors

Network processors can analyze the packet headers, implement look-up operations, and determine the output port of the packet very rapidly. The high throughput offered by the network processors is a result of their multi-thread operation and their multi-chip architecture. However, the processing of packets which belong to the same connection by different processors destroys the order of the packets. Hence, network processors cannot be used for the implementation of most scheduling algorithms. Also, in order to have proper scheduler architecture, the execution time of each operation in the processor should be well known by the designer. This makes it necessary to adopt the scheduling algorithm to the processor's command set. Unfortunately, the implementation result varies according to the processor architecture [37, 38].

3.1.1.4 ASIC

ASIC designs achieve the fastest and most successful results for the network applications. However, ASIC design is quite expensive and takes long time. After the design is completed, it is not possible to make modifications on the design [39]. Because of the fact that electronic technology grows very rapidly, ASIC cannot respond to the modification and maintenance needs in the network systems. As a result of these, ASIC designs are rather expensive and do not provide elasticity required for the network applications.

3.1.1.5 FPGA

FPGA designs do not need long design time as ASIC. FPGA technology provides a suitable design environment for the schedulers with many logic cells and high clock speed. The design modifications and improvements can be achieved easily on the FPGA. Also, FPGA design can be the first step to the ASIC design.

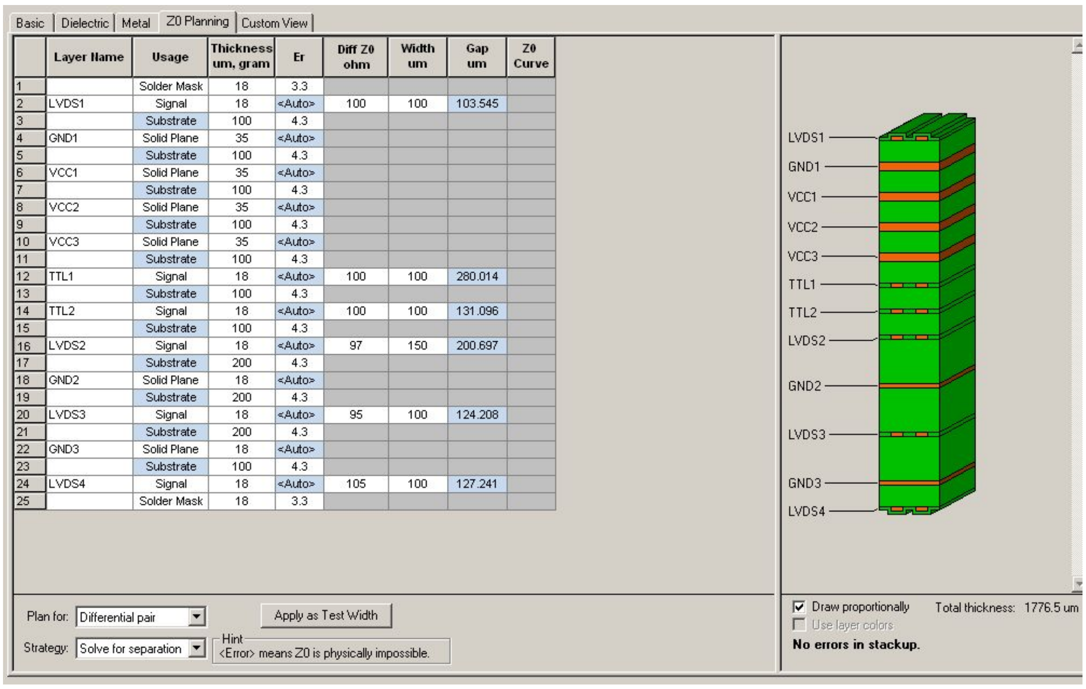
In the light of the given relative strengths and weaknesses of possible processor choices, FPGA is preferred for the scheduler implementations in this thesis. Our trial implementations showed that the FPGA should have more than 10000 cells and should support clock frequencies higher than 100 MHz. Also, the FPGA should have more than 400 user configurable I/O for memory interfaces and backplane connections. Xilinx Virtex2Pro20FF1152 is selected because of its widely tested architecture and rich library support. It has 20000 cells and 652 user I/O. An onboard crystal oscillator provides 125MHz clock signal to the FPGA. Intel 28F640 is selected as the flash memory because of its short access time and our available experience on the product. The platform flash memory is selected as Xilinx XCF16P. Visual Studio .NET platform is selected to communicate with the board through a graphical user interface.

3.2 HARDWARE DESIGN OF THE DATA-PROCESSING BOARDS

The hardware design started with the schematic design of the boards. The connections of the FPGA, other integrated circuits, power conversion and distribution circuitry, and clock circuitry are specified on the Mentor Graphics Design Architect software as huge sheets of logic design.

After schematic design, layout of the printed circuit board (PCB) is designed on the same software. The PCB is designed with special care on the spacing between the lines. There are more than 1000 signal lines connected to the FPGA and to be able to pass the ball grids, signal line width is selected as 5 mils. The PCB has 14 layers and a total thickness of 1.8 mm. Gerber files are simulated with HyperLynx software for signal coupling. The simulations showed that the coupling between any of the lines is no more than 300 mV. Figure 3-1 shows the PCB layers as seen on the HyperLynx software user interface.

The PCB is produced in ILFA, Germany. The basic building blocks of the data-processing board is given in Figure 3-2. Figure 3-3 shows the upper view of the data-processing board.



	Layer Name	Usage	Thickness um, gram	Er	Diff Z0 ohm	Width um	Gap um	Z0 Curve
1		Solder Mask	18	3.3				
2	LVDS1	Signal	18	<Auto>	100	100	103.545	
3		Substrate	100	4.3				
4	GND1	Solid Plane	35	<Auto>				
5		Substrate	100	4.3				
6	VCC1	Solid Plane	35	<Auto>				
7		Substrate	100	4.3				
8	VCC2	Solid Plane	35	<Auto>				
9		Substrate	100	4.3				
10	VCC3	Solid Plane	35	<Auto>				
11		Substrate	100	4.3				
12	TTL1	Signal	18	<Auto>	100	100	280.014	
13		Substrate	100	4.3				
14	TTL2	Signal	18	<Auto>	100	100	131.096	
15		Substrate	100	4.3				
16	LVDS2	Signal	18	<Auto>	97	150	200.697	
17		Substrate	200	4.3				
18	GND2	Solid Plane	18	<Auto>				
19		Substrate	200	4.3				
20	LVDS3	Signal	18	<Auto>	95	100	124.208	
21		Substrate	200	4.3				
22	GND3	Solid Plane	18	<Auto>				
23		Substrate	100	4.3				
24	LVDS4	Signal	18	<Auto>	105	100	127.241	
25		Solder Mask	18	3.3				

Plan for: Differential pair
Strategy: Solve for separation
<Error> means Z0 is physically impossible.
[✓] Draw proportionally
[ ] Use layer colors
Total thickness: 1776.5 um
No errors in stackup.

Figure 3-1 The PCB layers as seen on the HyperLynx software user interface.

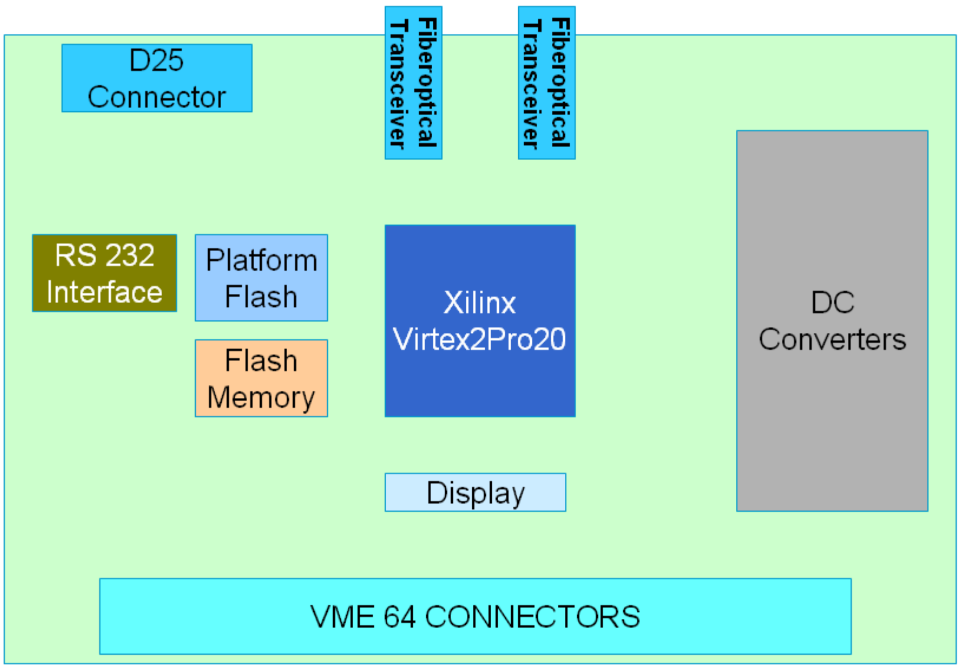


Figure 3-2 The basic building blocks of the data-processing board.

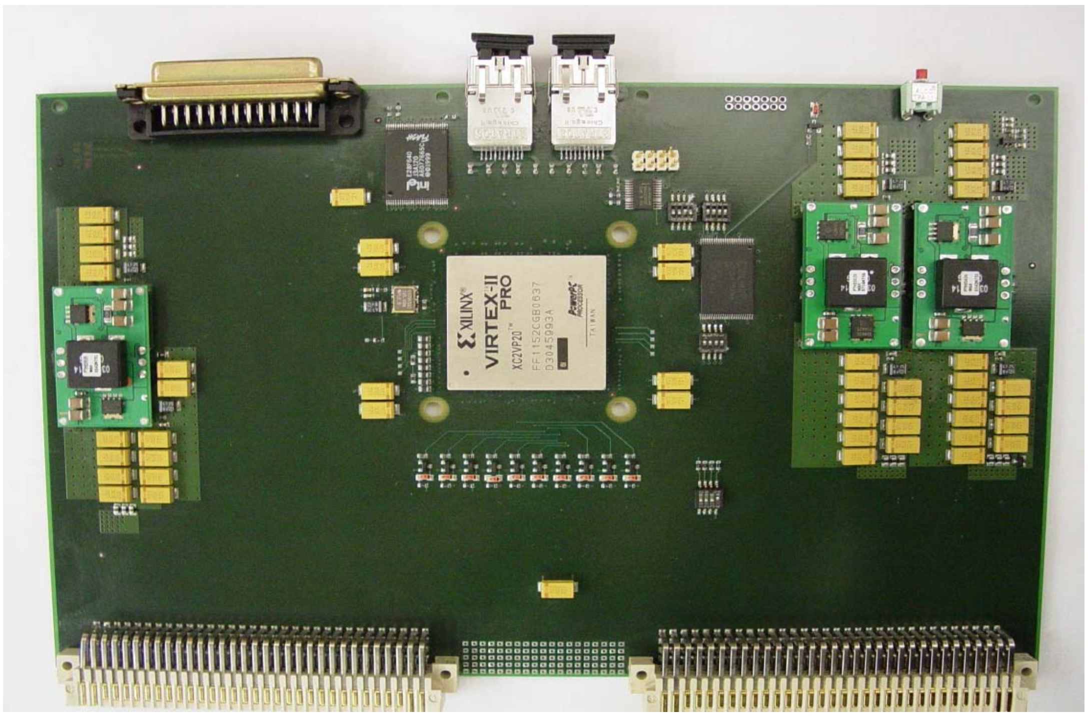


Figure 3-3 The upper view of the data-processing board.

3.3 HARDWARE DESIGN OF THE BACKPLANE

The backplane connects the data-processing boards with four 24-bit busses. Figure 3-4 shows the building blocks of the backplane. The bus lines are connected to D-25 connectors for further extension requirements and testing purposes. Also, the backplane distributes the 5V power to the data-processing boards. Same design steps are followed for the design of the backplane. The top level schematic view of the backplane design as seen on the Mentor Graphics Design Architect software is given in Figure 3-5.

After the production of the data-processing boards and the backplane, each board is tested for design and manufacturing errors. Following the tests, the data processing boards are integrated to the backplane to form the final testbed. Figure 3-6 shows the hardware testbed.

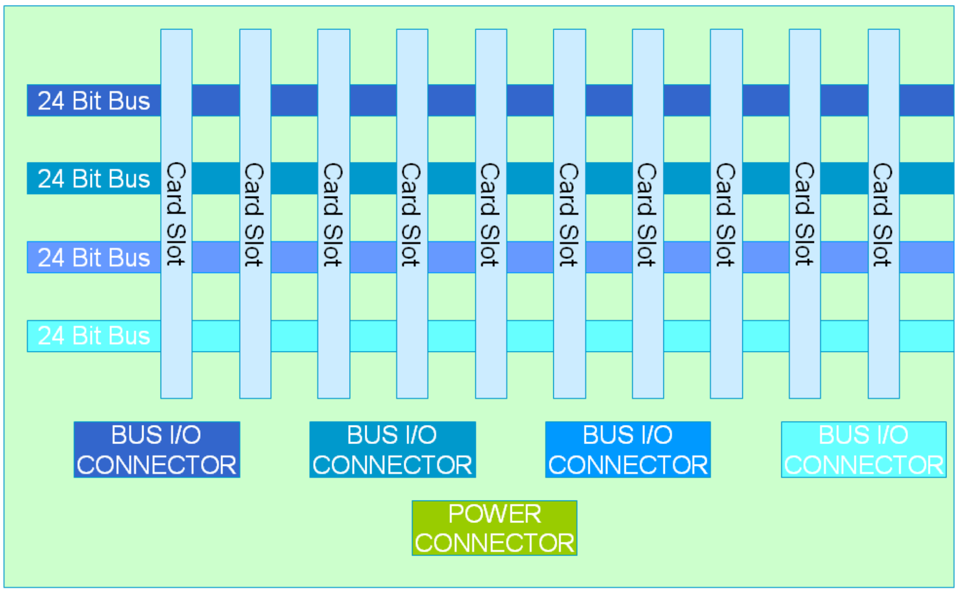


Figure 3-4 The building blocks of the backplane.

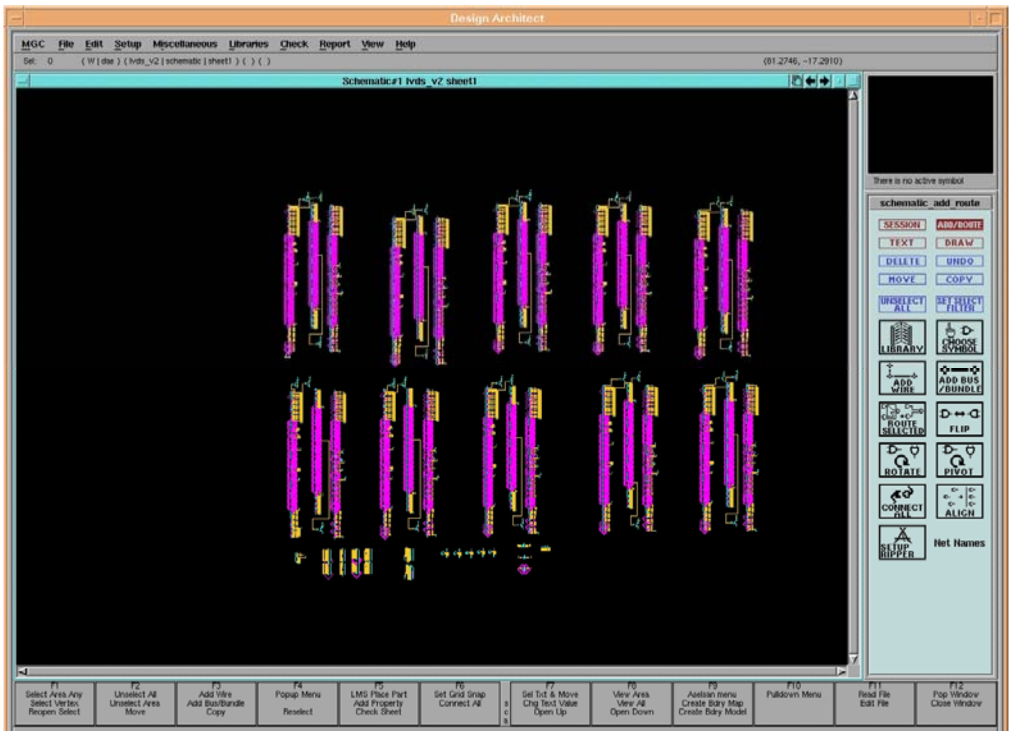


Figure 3-5 The top level schematic view of the backplane design as seen on the Mentor Graphics Design Architect.

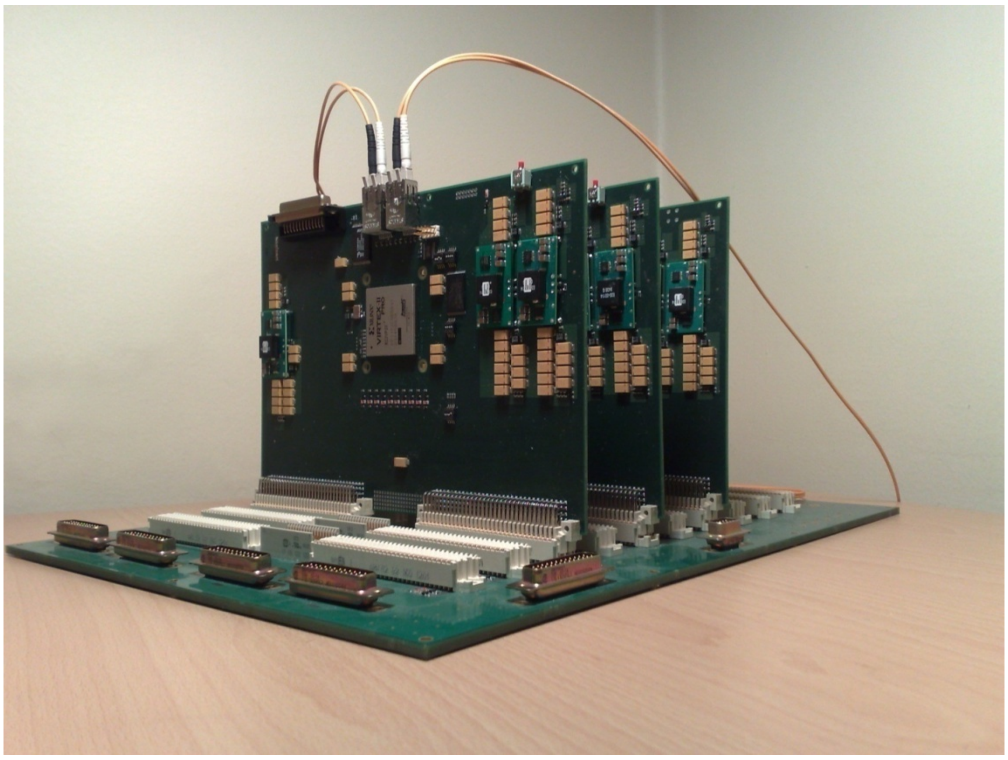


Figure 3-6 The hardware testbed.

3.4 LOGIC DESIGN WITH FPGA

FPGA is composed of thousands of configurable logic blocks (CLB). These logic blocks are configured during the implementation of the design according to the required hardware structure. The hardware behavior is first defined with one of the hardware description languages: VHDL or Verilog. Then, the system behavior is tested with one of the simulation tools: Xilinx ISE Simulator or Modelsim. These simulation tools have specific libraries for each family of FPGA devices and could perform realistic results. After the simulation, the design is implemented in hardware. In the implementation phase, the signals are assigned to the selected FPGA pins, time constraints are used to force the place and route process to optimize the length of the routes and placement of the design. After that, a BIT file is generated. This file contains all the information for the implementation of the design and defines the configuration of the logic blocks and the routing between the logic blocks. The BIT file is loaded onto the hardware using IMPACT software and Xilinx Parallel JTAG Programming Cable. The behavior of the system can be tested by observing the signals on hardware by using the software ChipscopePro. This software uses JTAG pins of the FPGA to monitor the signals and displays the results on PC with a graphical interface.

In this thesis work, VHDL is used for hardware description. Xilinx ISE 9.2 is used as code development environment. ChipscopePro 9 is used for hardware testing.

CHAPTER 4

FPGEN: A FAST, PROGRAMMABLE TRAFFIC GENERATOR

The increasing bandwidth and the variety of new applications of computer networks continuously motivate both academic and industrial research for the development of new network equipment such as switches and routers as well as new applications and protocols. In this respect, traffic generators are required to test and evaluate the performance of network applications, protocols, equipments or an entire network under predetermined load conditions. The packets can be generated with a traffic pattern according to a stochastic specification or based on a previously collected trace. Network equipment manufacturers use traffic generators to test their equipment in the laboratory environment and to demonstrate their capabilities to their customers. Benchmark tests are performed in evaluation labs to test and certify the equipments from different manufacturers by the help of very capable (and expensive) traffic generators [40-42].

There is a large number of academic studies on the design and evaluation of different switch architectures, fabric and QoS scheduling algorithms and buffer management strategies. Important performance metrics such as packet delay and loss depend on the management and scheduling of the buffers. These metrics are evaluated analytically by modeling the buffers as queuing systems with traffic arrivals such as Poisson, Bernoulli or Markov-modulated processes [43–52]. Hence, traffic generators which can produce packets according to these certain processes can demonstrate the accuracy of the analytical performance results when the proposed architecture is implemented in hardware.

The traffic generators are required to be scalable to high speeds (in bit and packet rates) and configurable to generate traffic according to the desired shape. While software-based traffic generators [53–60] can produce a wide variety of traffic patterns, they cannot reach high packet and bit rates [61]. Hence, design and implementation of high-speed packet generators that can generate the intended traffic properties on hardware is an important research issue.

The commonly used hardware platform for packet generator design in the previous academic literature is FPGA. Today's FPGAs comprise a very high number of logic gates and embedded blocks in small packages. When compared to full custom designs, FPGA technology enables the design of complex hardware platforms with reduced engineering cost and rapid turnaround time. Furthermore, FPGA-based prototype production is an important step for the verification of the expensive and time-critical ASIC projects [62]. It is possible to convert a hardware design on FPGA to ASIC provided that power source design, packaging and boundary scan testing constraints are taken into consideration [63].

Previous work on hardware packet generators features techniques that limit the scalability and flexibility of the design such as computing the packet generation times and packet sizes using on-board processors [64] and external computers [65], or relying on previously collected packet generation statistics at the expense of a memory access for each packet generation [66, 67]. Some of these previous studies are implemented and tested on hardware [64–67] while some of them are only simulated in software [68, 69]. Furthermore there is no evaluation of the generated traffic according to the desired statistics.

In this chapter, we present the design, implementation and performance evaluation of a hardware-based packet generator FPGEN (Fast Packet GENerator). FPGEN can generate Poisson traffic with exponentially distributed packet sizes and Markov modulated on–off traffic which are widely adopted traffic models. To this end, FPGEN can be used to evaluate the performance of switch fabric architectures, buffer managers and QoS support mechanisms such as schedulers and packet classifiers. The contributions of our research are as follows.

Firstly, our implementation is scalable to high-speeds as it is carried out purely on hardware without using any high level programming or processors. The packet generation times are computed in real-time entirely using the logic resources of the FPGA. FPGEN does not depend on any collected traffic trace and can be configured to generate traffic with different parameters exploiting the programmability of the FPGA. The hardware design of FPGEN can generate one packet per clock period per interface. This rate scales linearly with the number of interfaces and can be achieved for both Poisson and on–off traffic types. The FPGEN board has two interfaces and operates at 125 MHz which enables a maximum packet generation rate of 125 Million packets/second (pps) per interface and 250 Mpps total. FPGEN can fully utilize the two OC-48 fiberoptic interfaces and generate a maximum of 2.5 Gbps traffic per interface and a total of 5 Gbps. We provide the hardware implementation details and experiment results to demonstrate the capabilities of FPGEN. We did not find any previous work on hardware packet generators with such a detailed description of the design to justify the claimed packet and bit rates.

Generating traffic according to given statistics on a serial interface has inherent difficulties due to the required independence between the packet sizes and the inter-packet times. The second contribution of this work is presenting a model which can overcome these difficulties for Poisson traffic and be implemented on hardware. To the best of our knowledge there is no other published work on a hardware-based packet generator that produces Poisson traffic with exponentially distributed packet sizes.

Generation of Markov-modulated on–off traffic is studied before in [68, 69, 64]. However, the used techniques in the previous work are RAM-based and processor based. The third contribution of our work is applying our design approach to generate on–off traffic entirely on hardware.

Finally the fourth contribution of our work is the hardware design details, the experimental study carried out on hardware and its results that demonstrate not only the rate achieved by FPGEN but also the statistical properties of the generated traffic. [64–67] provide measurements of packet rate on hardware. However there is no presentation of the hardware design such as its state machine structure which can demonstrate the maximum number of clock periods to generate a packet. We provide the implementation details to justify that our design is capable of generating one packet per clock period per interface and this rate scales linearly with the number of interfaces. In addition, our experimental study shows that the interpacket times and packet sizes for the Poisson traffic and the average burst sizes and the load achieved for Markov-modulated on–off bursty traffic achieve the targeted statistical properties.

The remainder of the chapter is organized as follows. In Section 4.1, we summarize and discuss the previous work in the literature on traffic generators. We introduce the conceptual design followed by the hardware design and implementation of the Poisson traffic generation of FPGEN in Section 4.2. Furthermore, we demonstrate the generated traffic rates and their statistics. We present the design and evaluation of the Markov-modulated on–off traffic generation of FPGEN in Section 4.3. We summarize the features of FPGEN in Section 4.4, after demonstrating them by our experimental studies. Our conclusions are given in Section 4.5.

4.1 SYNTHETIC TRAFFIC GENERATION FOR THE PERFORMANCE EVALUATION OF COMPUTER NETWORKS

Performance evaluation studies in computer networking research require synthetic traffic generation. To this end, traffic generators are used to replicate the traffic conditions of the specific network environment that the device or the protocol under test will be deployed on. According to the device, component or protocol to be tested different parameters of the generated traffic are significant. While the packet rate is important to test a packet classifier or a packet scheduler, the load conditions, the inter-packet time and packet size distribution have to be considered to test a new buffer management algorithm. The validity of statistical approaches can only be justified with precise replication of the assumed traffic conditions.

4.1.1. Proprietary Hardware Traffic Generators and Software Traffic Generators

Traffic generators can be software or hardware-based. The hardware-based packet generators such as [41, 42] are usually professionally developed and purchased at expensive prices by network device manufacturers. These hardware packet generators can achieve high packet and data rates with different traffic profiles, however due to their cost and proprietary design they do not fit well into the academic networking research.

There are a number of software-based traffic generators which provide flexible configurable environments at low costs. However, the bit and packet rates and the statistical accuracy of the generated traffic depend on the hardware that the software runs on. Botta et al. [61] present a detailed recent study on the performance of software traffic generators. In this study four packet level traffic generators [57–60] ([60] is also described in [54, 55]) are selected according to their popularity.

The results of [61] are summarized in Table 4-1. It is observed that the software traffic generators fail to achieve the imposed packet rate starting from fairly low rates. It is also observed that beyond 500 Mbps with the minimum-sized packets, the throughput capabilities of the investigated generators saturate. Furthermore starting from even lower rates the inter-packet times of the generated traffic are found to deviate from the expected distribution. The reason for this behavior is stated as the lack of dedicated buffers as opposed to hardware implementations and the involvement of the CPU which is an expensive operation.

Table 4-1 The evaluation of the previous work on the software traffic generators [61].

Name, ref.	**Traffic type(s)**	**Kpps**	**Mbps**
TG, [57]	Constant, uniform, exponential, on/off	70	600
MGEN, [58]	Constant, exponential, on/off	70	600
RUDE/CRUDE, [59]	Constant	80	500
D-ITG, [60]	Constant, uniform, exponential, Pareto, Cuchy, normal, Poisson, gamma, on/off	130	500

4.1.2. FPGA-Based Hardware Traffic Generators

FPGA-based hardware traffic generators both exploit the programmability of the FPGA to provide flexible implementation and avoid the problems of the software traffic generators as discussed above. Furthermore generation of traffic at high rates demands for concurrency, bit-level parallelism, high operating frequency and short memory access time which can be provided by the FPGA platform.

Traffic generation with certain statistical behavior requires a random number generator which does not repeat itself for a sufficiently long time, state machine structures for bursty traffic generation and different queues to aggregate traffic or store packets before they are transmitted. It is possible to design hardware random number generators on FPGA with very long periods until they repeat. Furthermore FPGA provides appropriate infrastructure for state machine implementation and ready to use blocks such as FIFO queues.

Generating a packet includes a number of steps such as deciding the packet transmission time, its size, its header and payload content that are independent from each other. Different packet streams can be generated independently both for multiplexing to achieve certain aggregate behavior or for transmitting on different interfaces. While, implementing these steps on a processor results in the sequential and hence slow execution, the structure of the FPGA is very convenient for designing logic circuits which are working in parallel. This enables the designer to use high degrees of concurrency and thus shorten the total execution time. Another favorable feature of the FPGA is that the execution time of each operation on the hardware is well known by the designer. This is very important especially while generating traffic according to a specific distribution in real-time to test a networking device.

Other features that make FPGAs preferred platforms for the design of traffic generators are their affordability, short development time and flexibility thanks to their programmability. After the FPGA design, the generated design files can easily be converted to ASIC designs with small effort [62, 63].

We present a comparative evaluation of FPGA-based hardware traffic generators in the literature with FPGEN in Table 4-2. Related work dates back to 1996 motivated by the then high-speed ATM technology [69]. In this study, the maximum traffic generation rate is computed as 4.5 Gbps by simply multiplying the trigger rate of 12 MHz and 53 bytes per ATM cell. However, no experimental results are reported to demonstrate that the packets are indeed generated at the stated maximum achievable rate with the correct stochastic distribution. In addition, no discussion is provided on the capability of the state machine structure in the designs to generate a new packet in each clock period.

Table 4-2 The evaluation of the previous work on the hardware traffic generators. Sim: Simulation, HW: Hardware, Mpps: Million packets per second.

Ref., year	**Traffic type(s), packet size(s)**	**Platform, approach, implementation**	**Freq, max bps**
[69], 1996	Markov Modulated Bernoulli Process (MMBP), fixed size (ATM Cell, 53 bytes)	Altera MAX Plus II2 [37], state machine, sim	12 MHz, 4.5 Gbps
[67], 2002	Long-range dependent, 4 different packet sizes: 40, 256, 512, 1500 bytes	Altera FLEXlOK250E-1, time-series data stored in RAM is used to represent the transmitted traffic, HW	12 MHz, 4.5 Gbps
[68], 2005	Bernoulli, 2-state Markov modulated, packet sizes are not specified	Xilinx Virtex-4 FPGA [38], state machine, sim	20 MHz, BW is not specified
[64], 2006	Self-similar, Bernouilli, Markov-modulated, 5 different packet sizes: 40, 512, 600, 700, 1500 bytes	Altera EP1SGX40GF1020 Stratix GX, on board NiOS processor is programmed to generate packets, HW	155.52 MHz, OC-48
[66], 2009	Any traffic type, any size	Xilinx Virtex II Pro 50 FPGA, PCAP file that is loaded into SRAM is replayed, HW	Not specified, 1 Gbps per interface, total: 4 Gbps
[65], 2009	Any traffic type, only packet headers	Xilinx Virtex II Pro 50 FPGA, controls the timing of packets received from asoftware traffic generator, HW	33 MHz (PCI freq.), total: 1 Gbps (Limited by PCI bus)
[21] FPGEN, 2010	Poisson traffic, 2-state Markov modulated, 50 different packet sizes (min: 64 bytes, max: 1536 bytes) payloads can be created as needed.	Xilinx Virtex II Pro 20 FPGA, linear feedback shift register for generating random variables, HW	125 MHz, 125 Mpps and 2.5 Gbps per interface, total: 250 Mpps, 5 Gbps

In [67], the interarrival times for the packets are first stored in a 64 megabyte off-chip memory in time-series format and then loaded into an FPGA via a PCI interface. The size of the memory determines the time period that the packet generation process repeats. The time-series data has predetermined intervals of 1, 10, 100 and 1000 ms, and an OC-48 rate of 2.36 Gbps is reached for packet sizes that are larger than 512 bytes. The statistical correctness of the generated traffic is demonstrated by comparing the Hurst parameters of the original time-series data and the measured time-series data. The implementation platform does not work standalone and requires a PC to work with. Furthermore the authors do not explain if it is possible to generate the payload of the packets.

The arrivals are generated using probability values that are stored in RAMs in [68]. The correct operation of the packet generator is verified using stimulus written in System C, and cosimulated with the Verilog HDL implementation, using the Synopsys VCS-MX simulator. In the implementation of this design, triggers are generated instead of real packets. Hence, no packet size or maximum achievable data rate information is provided.

[64] is the most advanced and the fastest stage of a series of traffic generators with the same design approach that are developed by the same authors. In this approach, the traffic generator is coded in C and downloaded to an NiOS processor. The processor runs the Micro C OS II operating system. Whereby it has to be noted that the use of a processor instead of a pure hardware design limits the system performance. Five different packet sizes are supported.

[65, 66] present packet generators implemented on NetFPGA. NetFPGA is a general purpose networking platform accelerator designed as a PCI card to be plugged into a computer. It contains a Xilinx FPGA, 4 Gigabit Ethernet ports, Static RAM (SRAM) and Double-Date Rate (DDR2) Dynamic RAM (DRAM) providing the interfaces and certain hardware blocks specific to networking applications such as queue managers or packet capture components [72]. NetFPGA is neither

designed nor optimized for traffic generation. Traffic generation is an application for NetFPGA in addition to others such as buffer managing, packet classification and traffic monitoring. Generating packets according to a given profile is realized by replaying packets from a (Packet CAPture-PCAP) dump file [66] or transmitting the packets generated by computer on the Gbps interface [65].

In [66] packets are produced on the board according to the packet size, timing and payload information in a given PCAP file. The design consumes 83% of the logic slices on Virtex II Pro 50 which shows that it is very expensive to implement such a hardware on ASIC. Any type of traffic can be generated provided that the corresponding PCAP file exists. However, a simple test such as investigating the packet delays under different average packet sizes requires the existence of appropriate PCAP files rather than adjusting certain parameters and running tests. There are additional issues related to packet generation by replaying previously collected traffic. First of all, traffic captured on a link with certain properties such as capacity and respective buffer size of the router does not always lead to realistic results for some other link with different properties. Furthermore the closed loop behavior of TCP or any feedback-based protocol cannot be accurately captured by the PCAP files that are collected from past measurements. Finally, the PCAP file contains the packet payload information. As a result of this, the size of the file grows with the increasing number of packets and limits the number of packets that can be generated before loading the PCAP file again.

Precise Traffic Generator (PTG) [65] is another NetFPGA based packet generator which can be integrated to software based packet generation tools. In this approach, the packets are generated on some host computer, sent to the NetFPGA board over the PCI bus and then transmitted onto a Gigabit Ethernet interface. PTG's main objective is to control the transmission times of packets on the interface. The statistical correctness of the generated traffic is demonstrated by comparing the interarrival times of the generated traffic by PTG and a software based network emulator rather than comparing to a mathematical model. PTG's traffic generation rate is limited by the 32 bit, 33 MHz PCI bus, which has a bandwidth of approximately 1 Gb/s. As a result, only the packet headers are sent over the PCI bus and the payloads of the packets are generated on the NetFPGA as all zeros. This traffic cannot be used to perform network experiments which are sensitive to packet payload. Although the authors offer adding a number of predefined packet payloads in the future, the predefined nature of the packets may put some limitations on the scope of experiments that can be performed with PTG.

The new version of the NetFPGA board comes with 4 10 Gbps interfaces. However, there is no indication that the design in [66] will scale to generate traffic at these rates. PTG presented in [65] is limited by the PCI bandwidth and will not be able to utilize the high speed interfaces.

[73] is a special purpose traffic generator to provide a data flow identical to the data coming from the detector and readout boards in a particle experimentat CERN [74]. The hardware traffic generator is designed as a result of the scalability problems seen in the software simulator [75, 76] which was operating on a PC server. Traffic generator reads experiment data from some network storage and formats the data before sending it to a computing farm. A development board with a Stratix IV GX FPGA is selected for implementation. It is argued that the 10 GbE interface of the development board enables a traffic generation rate of 10 Gbps. However, no design details or implementation results are presented to verify the scalability of the design up to this data rate. Because of the specific purpose in the design of this traffic generator, it does not function as an Ethernet traffic generator. It is integrated into a special control system [77]. In its current state, it is not possible to use it as a standalone hardware traffic generator in the tests of the high speed network equipment.

4.1.3. Traffic Types Generated by FPGEN

FPGEN can produce Poisson traffic with exponentially distributed packet sizes and Markov-modulated on–off traffic. These are two widely studied traffic profiles which are also included in the generated traffic types of the software traffic generators that we discuss in Section 4.1.1.

Poisson traffic was the first analytical model and is widely studied due to its elegant analytical properties. [43, 44] can be mentioned among many other studies that analyze the performance of their proposed switch and buffer architectures under Poisson traffic. Furthermore FPGEN achieves Poisson traffic by multiplexing a large number of Bernoulli arrivals. [43, 45–48] consider Bernoulli arrivals as traffic models in their analyses for various switch, buffer and scheduler designs. Poisson traffic streams are also suggested for traffic probing for active measurements of delays on network paths [78]. This method is widely accepted and recent papers study the cases where using Poisson probes is appropriate [79].

Although studies after 1990s suggest that the Internet traffic is long range dependent and of self-similar nature, recently there are indicators that Poisson arrivals can be used once again to model the Internet traffic [80]. [81] states that, based on traces from backbone networks, at sub-second time scales, backbone traffic appears to be well described by Poisson packet arrivals. In [82], it is shown that on high-speed links, toward the core of the Internet, the traffic is composed of large numbers of connections which smooths out the the burstiness and traffic becomes similar to Poisson arrivals.

Markov-modulated on–off traffic models introduce the notion of state to determine the probability law of the traffic and can be used to model the queuing behavior of switches and routers under bursty multimedia traffic [83, 84]. One of the early studies that discuss on–off traffic models is the highly cited work of [85]. These models are verified for contemporary traffic profiles by [49, 50] which argue that the on–off packet-level model is an accurate model for IP traffic at the aggregate level, and for persistent TCP connections respectively. [51, 52] study the call and burst level behavior of different serviceclasses of traffic flows with on–off-bursty traffic. A very recent study [86] shows that the traffic in data centers exhibit on–off behavior.

Lastly, self-similar traffic is another popular and widely studied traffic model [87]. [88] show that a self-similar traffic source can be modeled as an aggregation of a number of on–off traffic sources. Hence the on–off traffic generation of FPGEN can serve as a basis for self-similar traffic generation.

4.2. FPGEN POISSON TRAFFIC GENERATION

4.2.1. Conceptual Design

A Poisson process with rate λ is a sequence of events where the number of events in any interval of length t is Poisson distributed with mean $\lambda.t$. A Poisson process is a continuous-time stochastic process which is frequently used to model the packet traffic in communication networks due to its nice and tractable analytical properties and its fitness to model the aggregate effect of a large number of individuals operating independently.

There are two important constraints for a hardware traffic generator which generates traffic with specific distributions for the inter-packet times and the packet sizes. The first constraint is the discrete time operation of the hardware; the second constraint is the requirement to serially transmit the packets over the physical interface.

In this work we generate Poisson arrivals with exponentially distributed packet sizes while satisfying these constraints. To this end, we implement an approximation of the Poisson process with a discrete time Bernoulli process. In this approach there are n independent traffic sources. Over each clock period each source generates a packet with a uniform probability p constituting a Bernoulli process with n trials where the probability of success of each trial is p. For sufficiently large n and sufficiently small p the Bernoulli process approaches a Poisson process with rate $\lambda = n.p$. [89] states that this approximation of the Poisson process is a good one if n is at least 20. We selected $n = 50$ for our implementation after a search for a suitable number for the traffic source count n with a simulation study. It is possible to further increase n, however the occupied logic area also grows with n and the accuracy of the approximation does not improve significantly. The increased logic size uses more FPGA resources and puts extra production costs for custom designs. The packet sizes are selected from a set of 50 discrete packet sizes. Similar to the number of traffic sources, the number of different packet sizes is determined with a simulation study to achieve the best accuracy with a number of packet sizes as small as possible.

The inter-packet times and the packet sizes are independent from each other in the mathematical model of our packet generation process. Hence, it is possible that the time between two consecutively generated packets is smaller than the time to transmit the first packet on the serial interface. When the packets are generated and serially transmitted on a physical interface, choosing the inter-packet times and the packet sizes from their corresponding distributions requires continuous adjustment of these parameters to either fit the packet transmissions in the gaps between consecutive packet generation times or to modify these gaps to accommodate the packet transmission times. Not only is this a complicated task but also such adjustments can lead to large deviations from intended distributions.

In our hardware design, we make use of Burke's Theorem [90] to tackle this problem for Poisson inter-packet times and exponentially distributed packet sizes. Burke's Theorem states that for a First Come First Served (FCFS) queuing system with a single server, if the arrivals are Poisson arrivals with rate λ and the packet sizes are exponentially distributed (forming an M/M/1 queue), then the departure process is also a Poisson process with rate λ. Accordingly, in our design, we first generate packets with exponentially distributed sizes according to the Poisson process that is approximated using a Bernoulli process. These packets are input to an FCFS queue and transmitted one by one on a hardware interface to constitute a Poisson process according to Burke's Theorem as seen in Figure 4-1. Although the selecetion of the inter-packet times and the packet sizes are independent from each other, as a result of this queuing there is a depencency between them in the output channel. As the traffic rate increases, queue occupancy also increases and the interpacket times become more dependent on the packet sizes.

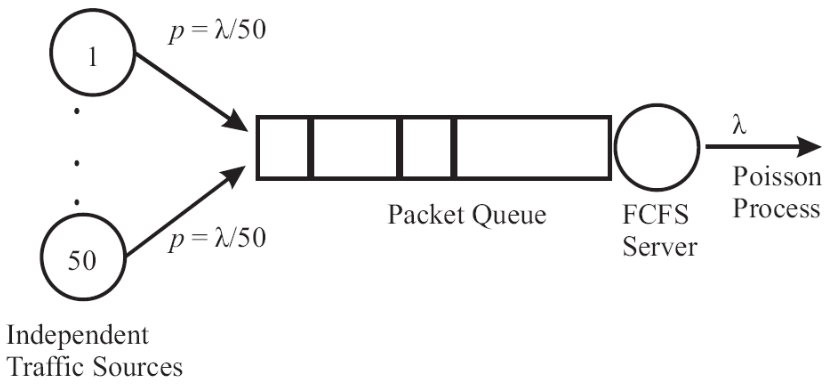


Figure 4-1 The design idea of the Poisson traffic generator.

4.2.2. Hardware Design

FPGEN is designed in the scope of our research for constructing a custom built high-speed network testbed. The FPGEN board is part of this testbed. However, it is possible to implement FPGEN on any board which contains enough FPGA resources and optical interfaces.

Figure 4-2 shows the basic building blocks of the FPGEN Poisson traffic generator. The traffic generator contains 50 traffic sources which generate packet-generated flags that are 1 bit pulses with the selected probability of $\lambda/50$ to achieve a certain Poisson arrival rate of λ. These flags are stored in the First Come First Served (FCFS) Flag Queue (FQ) with a maximum size of 1024 and processed by the controller unit one by one. The control unit determines the packet size and builds the payload of the actual packets corresponding to each flag. Once the packet is ready for transmission it is stored in the output buffer to be transmitted on the fiber interface immediately.

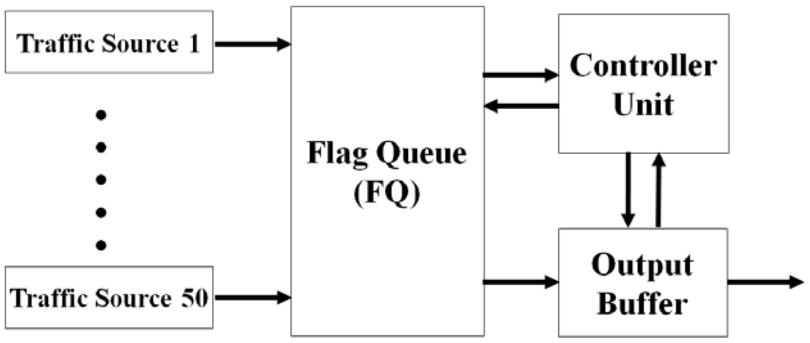


Figure 4-2 The basic building blocks of the traffic generator design.

The generation of packets with uniform probability by each traffic source is achieved by using 64 bit Fibonacci Linear Feedback Shift Registers (LFSR). A linear feedback shift register is a shift register whose input bit is a linear function of its previous state. The input bit is driven by the exclusive-or (XOR) of the selected bits (tap numbers) from the overall shift register content. The sequence of bits produced by the register is completely determined by its current state. The register has a finite number of possible states, consequently; it must eventually enter a repeating cycle. However, an LFSR with a well chosen feedback function can produce a sequence of bits which has a very long cycle and appears random.

When the outputs that influence the input (tap numbers) in a Fibonacci LFSR are selected from the coefficients of the non-zero terms of the appropriate primitive polynomials, the register cycles through a maximal number of states excluding the all-zero state [92]. In this work, 64 bit LFSRs are used. The LFSRs have 264 states and according to our calculations, this corresponds to a self-repeating sequence with a sufficiently long repeating period of 4680 years as shown below.

$$\frac{2^{64}\,states}{125Mstates/\text{sec}.(365.24.60.60)\text{sec}/\,year} = 4680\,years \qquad (4\text{-}1)$$

Each traffic source contains a 64 bit Fibonacci LFSR and two vectors of size 16 that we call reference vector and success vector. The LFSR in each traffic source is initialized by a different and randomly selected seed value. The seed is selected before the FPGA code is synthesized and is not modified afterwards. Hence, the LFSR in each traffic source generates a different pseudo-random bit sequence. The reference vector keeps a 16 bit subset of the 64 bits in the LFSR and is updated according to the content of the LFSR in each clock period. The success vector is used to determine whether a traffic source generates a packet. The success vector is preloaded with a 16 bit sequence n_{sv} according to the desired traffic load. In each clock period, the content of the reference vector is compared to that of the success vector. If the reference vector is smaller than the success vector, the packet generation attempt is successful and a packet-generated flag is asserted in that traffic source. Hence, each packet source generates packet-generated flags with a uniform probability of $n_{sv}/2^{16}$.

Whenever a flag is generated by any of the 50 sources, the flag is pushed into FQ that is shared by all of the traffic sources. The controller unit performs the traffic control procedure which includes monitoring the output buffer and FQ size constantly. If the output buffer is empty and FQ has a non-zero flag count, the top flag is popped. The controller unit generates the payload of the packet according to the randomly selected packet size information and puts the packet in the output buffer where it is transmitted immediately using the RocketIO Multi-Gigabit Transceiver [93]. No other flag is popped until the end of the packet transmission. If FQ is not empty at the instance of flag generation, this control procedure takes place while another packet is being transmitted and no packet delay is observed in the channel. However, in our simulations, we observed that when the queue is empty, the generated packet cannot be transferred to the output channel immediately because of clock periods spent when FQ is checked. We solve this problem by checking the channel if FQ is empty at the instance of flag generation, If the channel is empty, the packet is transferred to the output channel without visiting FQ. If the channel is busy, the flag is pushed to the queue. In this manner, the delay that takes place when FQ is empty is removed and the packet is directly transferred to the output channel.

The exponential distribution of the packet sizes is achieved by using a random-value vector and 49 boundary vectors of size 25 bits. The 49 boundary vectors partition the binary numbers from 0 to 225 into 50 segments. Figure 4-3 shows the boundary vectors and the segments.

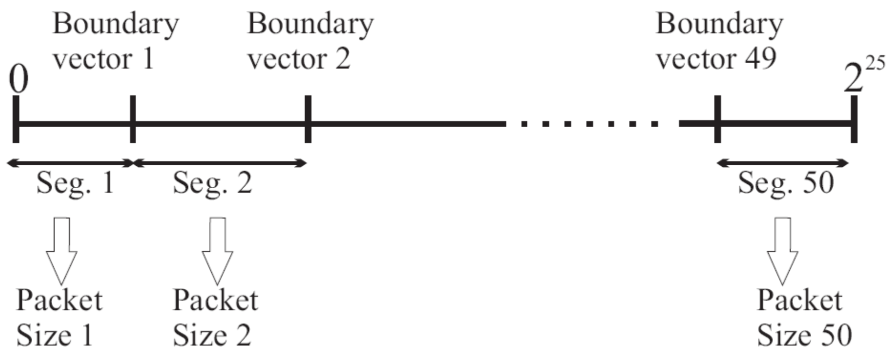


Figure 4-3 The boundary vectors and the segments.

The boundary vectors are calculated before the FPGA code is generated and are not modified later. The boundary vectors are selected such that the size of each segment is proportional to the probability of the generation of the corresponding packet size. The random-value vector contains a 25 bit pseudo-randomly generated sequence produced by the 64 bit LFSR's in the system. When a packet-generated flag is popped from FQ, the packet size corresponding to the segment which contains the random-value vector is selected as the size of the packet. The cumulative distribution of the packet sizes is shown in Figure 4-4.

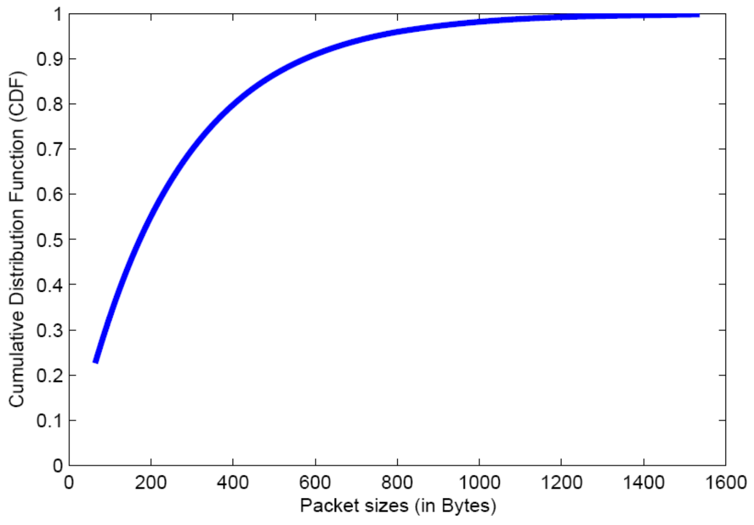


Figure 4-4 The cumulative distribution of the packet sizes.

The logic circuit updates the random value vector and compares its value with the boundary vectors continuously. Hence a new and random packet size value is calculated at each clock period. If a packet flag is present in a clock period, the current value of the packet size is selected as the size of the packet. If no packet flag is present in a clock period, the packet size value is not used for packet generation. As a result of this when a packet flag is generated, no additional time is required for the calculation of the size of the packet. When a packet flag is present in a clock period, the content of the packet is formed by simply copying related header and payload vectors which are present on the FPGA into the packet body. Each bit is copied in parallel, hence the content of the packet is also generated in a single clock period.

The selection of the packet size and formation of the packet content processes use very simple comparison and copying operations. As a result of this simplicity, one packet can be generated per interface, in each clock which enables generating at a maximum rate of 125 Mpps at 125 MHz. The sizes and contents of the packets on each interface will be independent from the ones on the other interfaces. Consequently the packet generation rate scales linearly with the number of interfaces and the supported clock frequency of the FPGA.

We collect the statistics of the packets generated by FPGEN to demonstrate the accuracy of the generated traffic with respect to the intended Poisson traffic shape. For this purpose, our design contains a 16 bit master counter which is incremented in each clock period and a 16 bit last packet transmission time vector which is updated at each new packet transmission. At the start of each new packet transmission, first the inter-packet time is calculated by subtracting the last packet transmission time vector from the value of the master counter at that instance. The calculated inter-packet time is stored in the flash memory. After that, the last packet transmission time vector is updated with the new master counter value. We transfer the inter-packet time data which is collected in the flash memory to a PC using an RS-232 serial interface.

The Poisson Packet Generator of FPGEN uses 38% of the available slices on the FPGA. The information about the utilization of the FPGA resources is presented in Table 4-3.

Table 4-3 The utilization of the FPGA resources for the Poisson traffic generator design.

Resource	**Used**	**Available**	**Utilization (%)**
Slices	3574	9280	38
4-Input LUTs	6653	18560	35
Flip flops	3537	18560	19
Block RAMs	0	88	0
External IOBs	36	564	6

4.2.3. Experiments and Performance Evaluation

The inter-packet times between consecutive packets for Poisson packet traffic with rate λ are exponentially distributed with a mean of $1/\lambda$. We aim to demonstrate how accurately the Poisson packet generator of FPGEN can follow the desired exponential distribution of the inter-packet times. To this end, we perform 6 experiments with different mean inter-packet times achieved by adjusting the success vector as described in Section 4.2.2. In all our experiments, there are 50 different packet sizes which are exponentially distributed between 64 and 1536 bytes as shown in Figure 4-4. The mean packet size is 265 bytes which corresponds to a mean sample packet time of $\mu =$ 132.5 clock periods with 16 bits/clock transmission speed. We collect 10000 inter-packet times and compute the average sample inter-packet time of $1/\lambda_i^s$ to determine the respective load $\rho_i = \lambda_i^s/\mu$ for each experiment i, i= 1–6. The load values and their corresponding observed rates per interface in Mpps and Gbps are presented in Table 4-4. Note that the traffic rate in Gbps includes the overhead for the fiber optic interface. It is observed that the traffic generation rate in Gbps can reach the full utilization of the OC-48 fiber optic interface. The packet generation rate exceeds 1 Mpps at this maximum rate.

Table 4-4 Experiment loads and data rates. The pps and bps rates are per interface.

Exp. i	$\boldsymbol{\rho_i}$	**Mpps**	**Gbps**
1	0.07	0.0825	0.175
2	0.15	0.165	0.35
3	0.30	0.33	0.7
4	0.60	0.66	1.4
5	0.80	0.89	1.86
6	1.0	1.1	2.5

Different from all of the other works in the literature, we not only state the generated traffic rate averages but also demonstrate that FPGEN generates traffic with the intended distribution. We present the cumulative distribution function (CDF) of the collected inter-packet time data in comparison to the computed CDF values for exponentially distributed data for visually demonstrating their statistical properties. In addition we employ the Kolmogorov–Smirnov Test (KS Test) which compares the distribution of a given data set to the hypothesized continuous distribution defined by the respective CDFs [94].

Let $x_{i,j}$ represent sample inter-packet time j (j = 1–10000) collected in experiment i (i = 1–6). We first compute the empirical CDF F_i^e for $x_{i,j}$ where $F_i^e(x_{i,j})$ is the ratio of inter-packet time measurements that is less than or equal to $x_{i,j}$ to all measurements in experiment i.

We also define the computed CDF $F_i(x_{i,j})$ values of $x_{i,j}$ as follows

$$F_i(x_{i,j}) = 1 - \exp{(\lambda_i^s . x_{i,j})} \tag{4-2}$$

for the exponential distributed inter-packet times. The respective F_i^e and F_i for each experiment i are plotted in Figure 4-5.

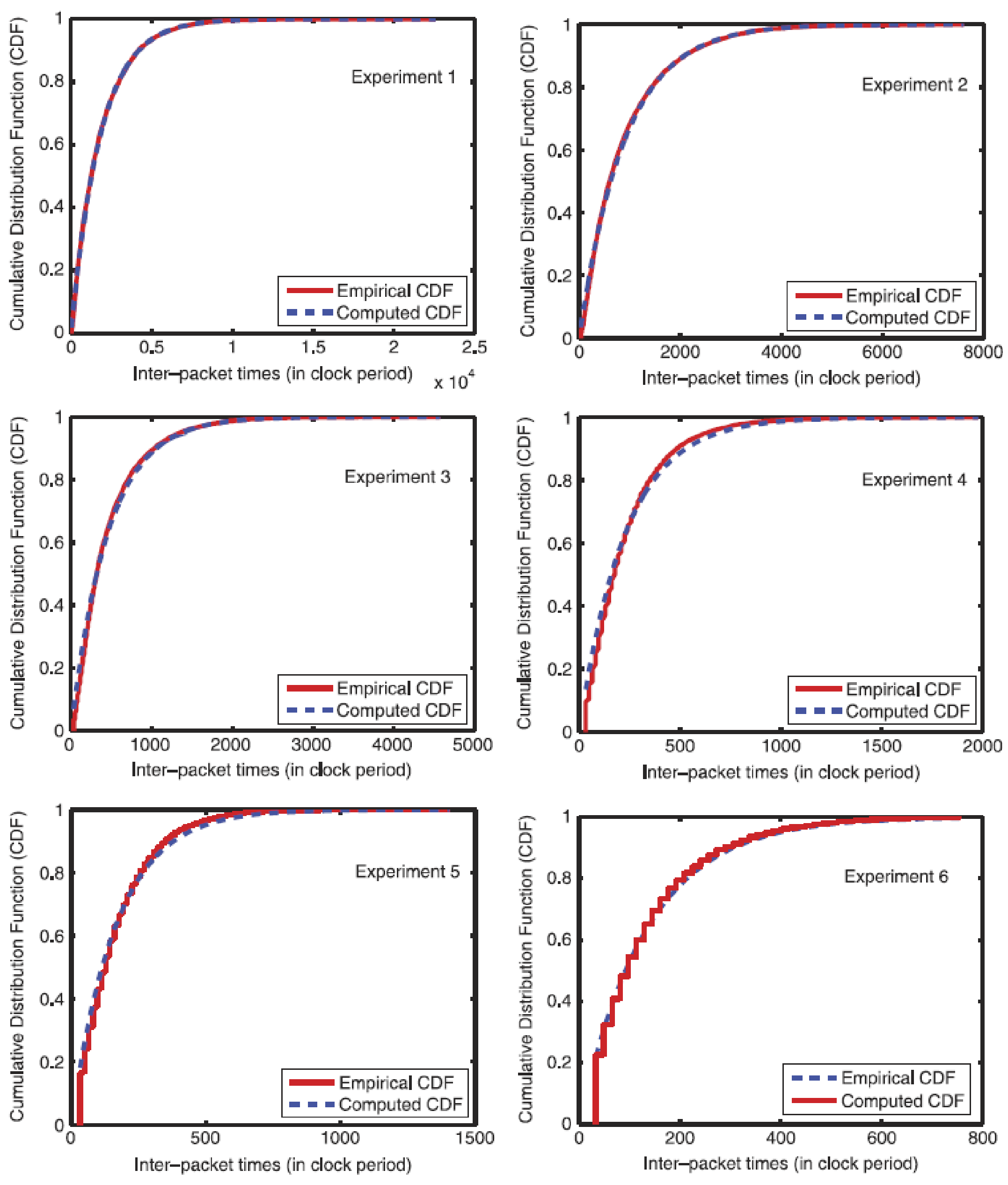


Figure 4-5 Experiment results for the comparison of generated traffic distribution and the theoretical expectation.

Our aim is to evaluate how closely the generated packet distribution F_i^e follows the intended distribution F_i. To this end, let

$$d_{i,j} = \frac{|F_i(x_{i,j}) - F_i^e(x_{i,j})|}{F_i(x_{i,j})}.100 \tag{4-3}$$

define the difference between the empirical and the computed CDF values for $x_{i,j}$. Then, we define d_i^{mean} and d_i^{max} as the mean and maximum values of $d_{i,j}$.

Our second evaluation is running the KS Test which compares F_i^e and F_i with a certain significance level for hypothesis testing. We use the kstest routine in the MATLAB [95] Statistics Toolbox. The default significance level is 0.05 (5%). We perform the test at significance levels of 1%, 2%, 5% and 10% where a significance level of 10% is the most strict test. We used randomly selected subsets of the original 10000 sample inter-packet times for the KS Test since even small deviations from the theoretical distribution are picked up by the test for large sample set sizes. The d_i^{mean} and d_i^{max} values with KS Test results for each experiment i are presented in Table 4-5.

Table 4-5 Differences between the empirical and computed CDF. Results of the KS Test. P: Pass, F: Fail.

Exp. i	$\boldsymbol{\rho_i}$	**CDF difference**		**KS Test results**			
		d_i^{mean} (%)	d_i^{max} (%)	%1 sig. level	%2 sig. level	%5 sig. level	%10 sig. level
1	0.07	0.39	2.20	P	P	P	P
2	0.15	0.77	4.51	P	P	P	P
3	0.30	1.23	6.77	P	P	P	P
4	0.60	1.91	8.84	P	P	P	F
5	0.80	1.84	8.73	P	F	F	F
6	1.0	1.08	6.32	F	P	F	F

Our experiment results show that for low ρ_i values (ρ_1 = 0.07, ρ_2 = 0.15, ρ_3 = 0.30) F_i^e follows F_i very closely. We have a large enough number of Bernoulli traffic sources that are multiplexed to generate the Poisson traffic. Hence the times for the packet generation events are Poisson distributed with the imposed average. In these low load ranges, the interpacket times are mostly larger than the duration of the packets. Thus, they are determined by the generation probabilities at the packet sources without a significant effect of the packet sizes. The KS Test yields Pass results for all of the significance levels.

The inter-packet times get shorter for a medium range load of ρ_4 = 0.60. This leads to an increase in the number of instances where the consequent packet generation happens before finishing the transmission of the current packet. In such cases the generation time of the consequent packet is delayed until the current packet transmission is finished which leads to the deviation of the statistics from the desired distribution. The KS Test yields a Fail result for the highest significance level of 10% for ρ_4 = 0.60.

The instances where the packet generation time is delayed due to the unfinished packet transmission start to dominate when the load becomes high (ρ_5 = 0.80, ρ_6 = 1). The inter-packet times start to be mostly determined by the packet sizes that is shown in Figure 4-4. The values that the inter-packet times can take are confined to the 50 packet sizes which explain the discrete look of the figure compared to the smooth curves for low load cases. ρ_6 = 1 indicates that there is always a ready packet to be transmitted. In this case F_i^e closely follows the exponential distribution of the packet sizes which is plotted in Figure 4-6. As the empirical CDF has a discrete form the traffic generation with ρ_5 = 0.80 passes only at the lowest significance level while ρ_6 = 1 fails completely due to the very discrete nature of the sample data.

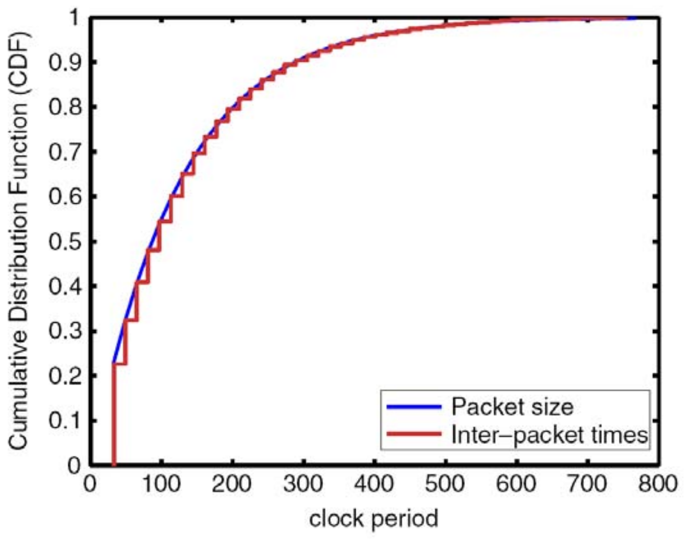


Figure 4-6 CDFs for packet sizes and inter-packet times at $\rho_6 = 1$.

Our experimental results show that packet generation process is fairly close to a Poisson process and the packet sizes are exponentially distributed as intended. To the best of our knowledge there is no other packet generator which achieves Poisson traffic at high bit and packet rates by only using the logic resources of the FPGA without any high level programming, processors or preloaded traffic data. CDF is a means of completely specifying the statistical properties of a set of data. Here we would like to note that, there is no other work that demonstrates the statistical properties of the generated traffic in comparison to the desired mathematical model using CDF including the software traffic generators such as D-ITG [60, 55] which are capable of generating Poisson traffic.

4.3 FPGEN BURSTY TRAFFIC GENERATION

4.3.1. Conceptual and Hardware Design

Next, we design and implement the FPGEN bursty traffic generator which generates Markov-modulated on–off bursty traffic to further demonstrate the capabilities of our FPGA-based traffic generator. FPGEN hardware is capable of generating packets of any size. We configure FPGEN to generate fixed packet sizes to be able to demonstrate how it can achieve the desired properties for given parameters.

Our bursty traffic source alternates between active and idle periods. It generates packets back to back during the active periods and stays silent during idle periods where the durations of active and idle periods are geometrically distributed. We employ a state machine structure with two states named active (A) and idle (I) as shown in Figure 4-7. In each state, there is a constant probability of switching to the other state. Let p be the probability of leaving the active state and q be the probability of leaving the idle state.

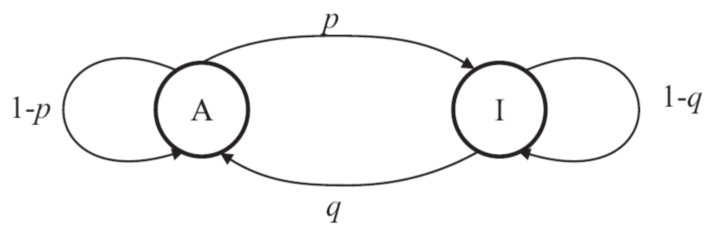


Figure 4-7 The state transition diagram of the bursty traffic generator design and the probabilities corresponding to the state transitions.

The probability that the length of the active period is i packet times is calculated as follows;

$$\Pr\{\text{Active period}=i \text{ packet times}\} = p(1-p)^{i-1} \tag{4-4}$$

which leads to a mean burst length of

$$\beta = \sum_{i=1}^{\infty} p(1-p)^{i-1} i = \frac{1}{p} \tag{4-5}$$

Similarly, the mean idle period is calculated as $1/q$. Let ρ denote the offered load defined as the ratio of the average time the system generates packets to the total time as:

$$\rho = \frac{\frac{1}{p}}{\frac{1}{p} + \frac{1}{q}} \tag{4-6}$$

Then for a given offered load ρ and mean burst length of β packets, the state transition probabilities can be calculated

$$p = \frac{1}{\beta} \text{ and } q = \frac{\rho}{\beta(1-\rho)} \tag{4-7}$$

In order to test the performance of the traffic generator on hardware, we implemented the design on our FPGA based board. In our implementation, constant state transition probabilities are generated using an LFSR in exactly the same way that was used in the Poisson traffic generator design. A single source is sufficient to generate the Markovian traffic as explained above.

We call the time that is required to transmit one fixed size packet a slot. The bursty traffic generator design contains a 64 bit Fibonacci LFSR and three vectors of size 16 that we call reference vector, active_to_idle vector and idle_to_active vector. The reference vector keeps a 16 bit subset of the 64 bits in the LFSR and is updated according to the content of the LFSR in each clock period. Each of the active_to_idle vectors and idle_to_active vectors is preloaded with a 16 bit sequence according to the desired state transition probabilities p and q as described above.

If the packet generator is in the idle state, at the end of each slot, the content of the reference vector is compared to that of the idle_to_active vector. If the reference vector is smaller than the idle_to_active vector, a state transition occurs. The traffic generator moves from the idle state to the active state. If the reference vector is greater than the idle_to_active vector, the traffic generator remains in the idle state. Similarly, if the packet generator is in the active state, at the end of each slot, the content of the reference vector is compared to that of the active_to_idle vector. If the reference vector is smaller than the active_to_idle vector, a state transition occurs. The traffic generator moves from the active state to the idle state. If the reference vector is greater than the active_to_idle vector, the traffic generator remains in the active state.

Hence, state transition occurs with a uniform probability. As long as the packet generator is in an active state, a fixed sized packet is generated in each slot. The packet content is generated in the same way as in the Poisson traffic generator. Hence, one packet can be generated in each clock period for each interface at 125 MHz. It is possible to generate bursty traffic on many interfaces by simply selecting different bits of the LFSR as the success vector for each interface.

The bursty packet generator of FPGEN uses 3% of the available slices on the FPGA. The largest use of the slices is for the implemented state machine architecture. The information about the utilization of the FPGA resources is presented in Table 4-6.

4.3.2. Experiments and Performance Evaluation

The packet size is selected as 8 bytes and the mean burst length is selected as 32 packets. This implies that p = 1/32. We tested the traffic generator in 3 load conditions of 1/3, 1/2 and 2/3. For these loads, the corresponding q values are calculated as 1/64, 1/32 and 1/16. Table 4-7 shows the test results for bursty traffic generator together with the calculated p and q values and the desired traffic conditions. The number of collected inter-packet times for each experiment is 10000.

Table 4-6 The utilization of the FPGA resources for the bursty traffic generator design.

Resource	**Used**	**Available**	**Utilization (%)**
Slices	339	9280	3
4-Input LUTs	447	18560	2
Flip flops	341	18560	2
Block RAMs	1	88	1
External IOBs	36	564	6

Table 4-7 Test results for bursty traffic generator together with the calculated p and q values and the desired traffic conditions. The pps and bps rates are per interface. Burst length is in number of packets.

Exp. i	**Desired**		**Calculated**		**Test results**			
	β_i	ρ_i	p	q	β_i	ρ_i	**Mpps**	**Gbps**
1	32	0.33	1/32	1/64	32.83	0.37	11.56	0.925
2	32	0.5	1/32	1/32	32.18	0.54	16.88	1.35
3	32	0.66	1/32	1/16	32.40	0.68	21.25	1.7

4.4. TRAFFIC GENERATION CAPABILITIES OF FPGEN

FPGEN can generate Poisson traffic with exponentially distributed packet sizes and Markov-modulated on–off bursty traffic. These traffic types are popular for modeling network traffic in a number of studies. Furthermore they are still valid models as we discussed in Section 4.1.3.

Our FPGEN board can generate a maximum of 125 Mpps per interface with its 125 MHz clock frequency. The maximum achievable data rate in bps further depends on the number of bits the interface can send per clock period. The interface on the FPGEN board can send 16 bits/clock period which enables the full utilization of the OC-48 fiber-optic interface and generates 2.5 Gbps per interface including the physical layer overhead. The total traffic generated by FPGEN can reach to 250 Mpps and 5 Gbps with its two interfaces. The pps rate achieved at a certain bps rate depends on the packet size. Our experimental results in Tables 4-5 and 4-7 show that FPGEN can achieve 2.5 Gbps per interface. The traffic statistics presented in Figure 4-5 and Table 4-7 demonstrate that FPGEN can achieve the intended statistical properties.

Testing certain router functionalities such as packet classification requires high pps rates rather than bps data rates. FPGEN can be used to generate the minimum size IP packets of 20 bytes length. In that case FPGEN can generate 25 Million IP packets/second with the interface speed of 16 bits/clock period. This rate can be increased if partial IP headers with smaller byte counts are adequate for the experiment or if our design is ported to another hardware platform with higher interface speeds.

Both the Poisson and on–off bursty traffic is generated using random processes that are implemented only by using the logic resources of the FPGA. The random elements in FPGEN operation such as inter-packet times, on–off state changes or packet sizes do not repeat for very long time intervals as described in Section 4.2.2. Note that FPGEN is the only hardware traffic generator that can generate Poisson traffic.

Here we would like to note that the rate of traffic that is generated by software traffic generators are far below FPGEN's rates as discussed in Section 4.1.1.

In comparison to previous work on hardware traffic generators, FPGEN stands out with its design that can generate one packet per clock period per interface as described in Section 4.2.2. This rate can separately be achieved for both Poisson traffic and on–off Markov-modulated traffic. Not only is there no current FPGA-based traffic generator with a higher rate than FPGEN but also its design can be ported to FPGA environments with a larger number of interfaces, faster interface and clock speeds to achieve rates that linearly increase with these parameters. To the best of our knowledge there is no previous work on hardware traffic generators that provides enough design detail and explicitly states how many clock periods it takes to generate a packet or how the design scales with the clock rate, the number of interfaces or interface speed.

Furthermore unlike [66] or [67], FPGEN does not depend on files that contain packet information. Although such design enables the packet generator to generate different traffic profiles according to the available file, adjusting traffic parameters requires appropriate files limiting the scalability of the approach. Unlike [64-66], FPGEN does not need any external hardware resources such as an embedded processor or a computer accessed via a PCI interface which limit the generated data rate and the scalability of the design.

FPGEN is able to generate any selected packet size distribution simply by modifying the content of the success vector. In addition to the 50 packet sizes which are currently available in the design, new packet sizes can be added by simply modifying the constant vectors in the design. The content of the packets can be specified by a C# based GUI running on a PC connected to the FPGEN board through the available RS-232 interface. It is possible to define fixed header fields and random fields in the packet payload that will be generated on the FPGEN board. Also the traffic load, success and boundary vectors can be modified using the same GUI.

4.5 CONCLUSIONS

Testing and performance evaluation of high-speed network equipment requires high-speed packet generators which can generate network traffic at predetermined load conditions and traffic patterns. Although they are very versatile, the software-based traffic generators do not scale to high speeds in the order of Gbps. Furthermore the lack of dedicated hardware resources and CPU-based operation lead to deviation from intended traffic profile at lower speeds than the saturation point of the throughput. Hence, the development of hardware-based traffic generators which can generate the imposed traffic characteristics is required for investigating the performance of backbone network devices according to metrics such as fabric throughput, buffer occupancy or QoS support.

In this chapter we present the design, implementation and experimental evaluation of a novel hardware-based packet generator, FPGEN, developed on FPGA. FPGEN is scalable to high-speeds as it is implemented purely on hardware without using any high level programming or processors. The packet generation times are computed in real-time entirely using the logic resources of the FPGA. FPGEN can generate one packet per clock period, hence it supports up to 125 Mpps per interface at 125 MHz clock rate of our board. Furthermore this rate scales linearly with increased clock rate, number of interfaces or interface speed. Our experiments show that the FPGEN board can support a total traffic generation rate of 5 Gbps and 250 million packets per second with its two OC-48 interfaces. FPGEN is configurable to generate traffic with different parameters due to the programmability of the FPGA. In this work, we present the design and implementation details of FPGEN followed by an experimental demonstration of achieving packet generation at OC-48 rate per interface.

FPGEN generates Poisson traffic with exponentially distributed packet sizes. We present a model which overcomes the inherent difficulties of generating this traffic on a serial interface due to the required independence between the packet sizes and the inter-packet times. In addition, FPGEN can generate Markov-modulated on–off traffic entirely on hardware. The above mentioned traffic rates can be achieved separately for both Poisson and on–off traffic.

Different from previous studies, we provide the implementation details to justify that our design is capable of reaching our claimed rates. Furthermore we demonstrate that FPGEN can generate traffic at these rates with the intended statistical properties by hardware experiments.

It is possible to incorporate other traffic generation patterns such as self-similar traffic in FPGEN. A self-similar traffic source can be modeled as aggregation of a number of on–off traffic sources where the the burst size is distributed according to Pareto distribution. Such burst sizes can be achieved by a similar procedure to our determining the packet size. We can reuse our uniform number generator and simply adjust the segment sizes of the boundary vectors to determine the burst lengths. Next, the packets generated from these sources can be aggregated using the same FCFS queue structure that we use for the Poisson traffic.

FPGEN serves as the packet generator for the performance evaluation of the developed schedulers.

CHAPTER 5

HARDWARE DESIGN AND IMPLEMENTATION OF PACKET FAIR QUEUING ALGORIHMS

The scheduling algorithms which are used in the QoS schedulers of the routers and the switches play an important role in determining the QoS performance of the Internet. In order to provide QoS support, the scheduling algorithms specify the order in which the packets queued at the output ports is actually transmitted. The scheduling algorithms give different service to different flows.

A class of PFQ algorithms emulates the behavior of the ideal GPS scheduling [7-17] which cannot be used in packet switching networks. In all PFQ algorithms, a global function called virtual time is used to track the progress of the GPS scheduler. For each head of line (HOL) packet of each flow in the system, a finish time is calculated. This finish time corresponds to the time that this packet would leave the GPS scheduler. Packets are served in the order of their respective finish times. The finish time of a packet is the sum of its start time and the time needed to transmit the packet. The start time corresponds to the time that this packet would start receiving service in the GPS scheduler.

In this chapter, we propose a hardware architecture for the design of the general family of PFQ schedulers. We define the blocks that are common in all PFQ schedulers and the blocks that are unique to each scheduling algorithm. In this architecture, we identify the design challenges and use techniques to overcome these difficulties. We use our proposed architecture to implement the WF^{2}Q+ algorithm. The algorithm is implemented on a FPGA based board and the performance evaluation is performed on a hardware testbed.

The remainder of the chapter is organized as follows. In Section 5.1, we summarize and discuss the design of a dynamically adaptable PFQ scheduler. We review the previous work on the hardware implementation of PFQ schedulers in Section 5.2. We introduce the hardware design followed by the implementation of the WF^{2}Q+ scheduler in Section 5.3. In Section 5.4, we present the performance measurement of the dynamically adaptable WF^{2}Q+ scheduler. Our conclusions are given in Section 5.5.

5.1 DESIGN OF A DYNAMICALLY ADAPTABLE PFQ SCHEDULER

5.1.1 PFQ Schedulers

In a PFQ scheduler, there are N queues corresponding to N flows. Each queue i has a minimum bandwidth allocation r_i (i= 1,..., N). The service share of each queue is proportional to its r_i. In the scheduler, there is a global virtual time function $V(t)$. This function is used to represent the progression of the simulated GPS scheduler. For each queue, there is a virtual finish time function $F_i(t)$ and a virtual start time function $S_i(t)$. The service order of the packets is determined according to the order of the packets' finish times. The packets get service starting from the one having the smallest finish time.

In all the schedulers belonging to the family of PFQ schedulers, $F_i(t)$ and $S_i(t)$ are calculated similarly. The main difference among these schedulers is in the calculation of the $V(t)$. For each queue i, $F_i(t)$ and $S_i(t)$ are updated in only 2 cases:

1. A previously empty queue has an incoming packet that immediately becomes head-of-line.

$$S_i(t) = \max(F_i(t\text{-}), V(t)) \tag{5-1}$$
$$F_i(t) = S_i(t) + L_i / \mathrm{r_i} \tag{5-2}$$

2. In a non-empty queue, a packet is departed and the next packet becomes head-of-line.

$$S_i(t) = F_i(t\text{-}) \tag{5-3}$$
$$F_i(t) = S_i(t) + L_i / \mathrm{r_i} \tag{5-4}$$

where $F_i(t\text{-})$ is the finish time of the queue i before the update and L_i is the length of the head-of-line packet for queue i.

5.1.2 Block Level Design of a PFQ Architecture

Considering the common tasks accomplished in the PFQ schedulers, we formed a block level architecture which is shown in Figure 5-1. The general blocks consist of a packet reception module, start and finish time calculators, a packet selection module, a counter aging module and a packet transmit module. The algorithm-specific blocks contain the $V(t)$ calculator and an eligibility checker for certain PFQ algorithms.

Each received packet enters the scheduler through a packet reception module. According to the flow identifier in the packet header, the packet is delivered to the corresponding queue. For each queue, there is a start time calculator and a finish time calculator. The start time calculator module calculates the start time of the HOL packet in the queue according to (5-1) and (5-3). The finish time calculator module calculates the finish time of the HOL packet in the queue according to (5-2) and (5-4). If the system is not idle (not all the queues are empty) and a packet is not being transmitted, the packet selection module selects one of the HOL packets for transmission. Specific to the algorithm, the packet selection module works with an eligibility checker such that only packets whose start time is not greater than the current value of the virtual time are selected. For the selection of the packet, the queue with the minimum finish number is searched. A searching module is used to find the minimum of a group of numbers. Virtual time calculator uses a function which is unique to the scheduling algorithm to calculate the virtual time of the system. Depending on the PFQ algorithm, this function may include maximum or minimum operations. Hence, a searching module may also be requireded in this module. Packet transmission module transmits the packets which arrive from the packet selection module. Due to the continuous increase in $V(t)$, $F_i(t)$ and $S_i(t)$ vectors, it is expected that after some time, these vectors overflow and restart counting from 0. A counter aging module is used to prevent them from overflowing.

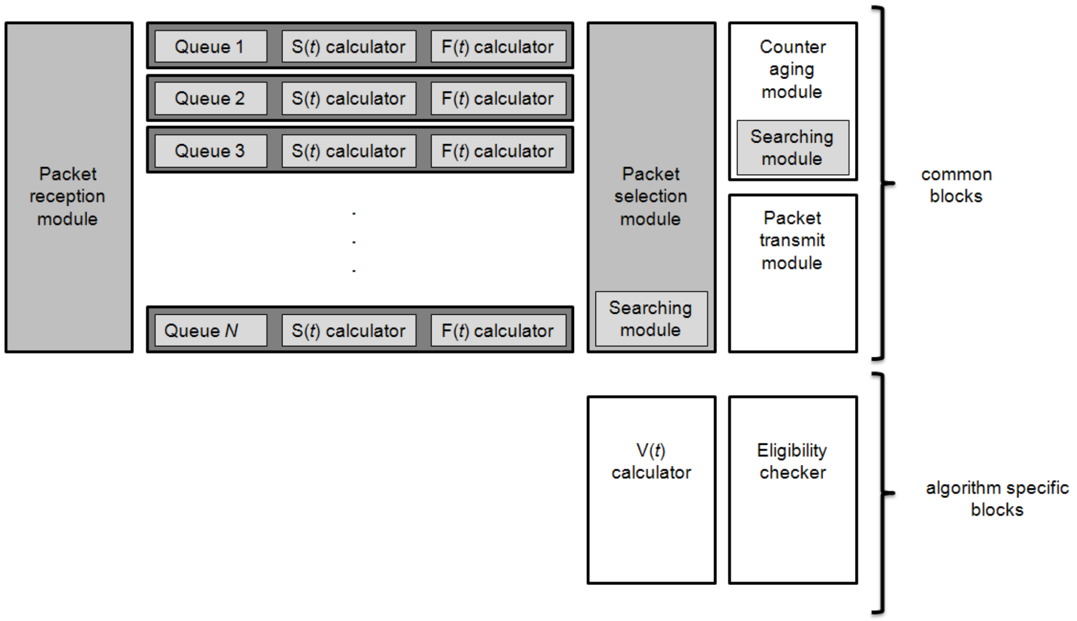


Figure 5-1 The block level architecture of the PFQ schedulers.

5.1.3 Design Challenges and Proposed Solutions

The QoS schedulers are generally implemented in hardware to support wire speed operation. Ideally the operation of the scheduler has to be completed in one clock cycle as it is possible that new packets can arrive at head-of-line position in each clock cycle. Hence, pipelined designs that aim to increase throughput with a trade off in delay might lead to a large number of packets that cannot get the agreed sevice.

The implementation complexity of the PFQ algorithms comes from two components. The algorithm specific component includes the calculation of the $V(t)$ and, if relevant, the eligibility check operation for a given PFQ algorithm. The general component consists of carrying out the updates for $S_i(t)$ and $F_i(t)$ as presented above and searching for the non-empty per-flow queue whose head-of-line packet has the minimum finish time.

The relevant design and implementation problems for the general component can be listed as follows:

5.1.3.1 Hardware Division

$F_i(t)$ is calculated for each queue in the scheduler. The calculation of $F_i(t)$ function requires the division of the L_i with r_i. Hence, the system design requires a lot of hardware dividers working in parallel which results in the waste of a huge amount of logic resources. We choose to prevent the waste of resources by using a look-up-table instead of dividers. The basic structure of the table is given in Table 5-1.

The first column of this table contains the possible L values in the network in ascending order. Let the scheduler support R_{count} distinct rates where r_{base} bps denotes the greatest common divisor of these rates. We divide all rate values by r_{base} to achieve the normalized set of rates where the minimum rate is r_{min} and the maximum rate is r_{max} (both without units). The first row of the table contains all of the featured normalized rate allocations from r_{min} to r_{max} in ascending order. Then, the cell of the table located at row i and column j contain the result of the corresponding L_i / r_j operation.

Table 5-1 The basic structure of the look-up table which is used in the division operation.

	$\mathbf{r_{min}}$	**…**	$\mathbf{r_j}$	**…**
$\mathbf{L_{min}}$	L_1 / r_1	…	L_1 / r_j	…
…		…	…	…
$\mathbf{L_i}$	L_i / r_1	…	L_i / r_j	…
…		…	…	…

In order to reduce the number of bits used in the calculation of $F_i(t)$ and $S_i(t)$ and to prevent the aging of the counters early, we need to have as small L_i / r_j values as possible. We achieve this by scaling the contents of the table with minimum L_i / r_j value found on the table. The minimum value is obtained from the division of the minimum L_i value with the maximum r_j value and found on the upper right corner of the table.

In order to divide L_i by r_j, it is enough to read the content of the cell which is found on the intersection of the row i and column j.

5.1.3.2 Dynamical Adaptation

We keep two separate look-up tables for division. In the first table, r values from r_{min} to r^{1}_{max} are used. In the second table, all of the r values are used. When there are no flows present in the system with r values higher than r^{1}_{max}, the first table is used. Whenever a packet is received from a flow which has a r value greater than r^{1}_{max}, the scheduler dynamically starts using the second table.

In the first table, the maximum r is r^{1}_{max} and in the second table, it is r_{max}. Consequently, the maximum L_i / r_j value in the first table is several times larger than the maximum value in the second table. As a result of the fact that the table is scaled with the maximum L_i / r_j value, the first table contains L_i / r_j values which are quite smaller than the corresponding entries in the second table.

Whenever the flows with high r values are present in the scheduler, they will get very high share on the service and as a result of this, they will leave the scheduler in a very short time. This will enable the scheduler to use the first table most of the time. Hence, using a table with smaller entries will slow down the aging of the $F_i(t)$, $S_i(t)$ and $V(t)$ counters in the scheduler.

5.1.3.3 Counter Aging

As a result of the continuously increasing behavior of $F_i(t)$, $S_i(t)$ and $V(t)$ vectors, after some time, these vectors reach their maximum (all bits are ones) and then, they overflow and restart counting up from zero. This problem is known as counter aging problem. As a solution to this problem, we apply a counter renewal procedure.

For v bit vectors, the maximum value that can be represented is $V_{max} = 2^{v}$. When the value stored in any of these vectors reaches a renewal threshold, such as $V_{max} \cdot 0.75$ then the values that are stored in all of the vectors are decreased by the minimum of $S_i(t)$. Note that the definition of $F_i(t)$ in (5-2) and (5-4) guarantees that the minimum value to be subtracted will not be stored in an $F_i(t)$ vector. The relative ordering of packet departures are not affected as all the vectors are decreased by the same amount. Such renewal process is completed in one clock cycle.

5.1.3.4 Searching

In the scheduler design, it is necessary to find the minimum of the time stamps in several different places. When selecting a packet for transmission, it is necessary to search the minimum $F_i(t)$ among all the queues. Also, in our solution to the counter aging, again it is necessary to find the minimum $S_i(t)$. As a result of the fact that the searching is needed in several places in the design, searching efficiency in terms of speed and logic area affects the performance of the scheduler closely.

In our architecture, a fast searching method such as RAM-based searching engine [96] should be used for large number of flows. This search engine uses a calendar queue data structure [97]. In this data structure, a priority queue is used to keep the timestamps of all the HOL packets presorted. A hierarchical searching mechanism [98] is performed on the priority queue to find the smallest timestamp quickly and efficiently. A simple comparator tree can be used as well when the number of flows in the design is small. A pipelined implementation such as [19] would slow down the operation of the scheduler.

5.2 REVIEW OF THE PREVIOUS WORK ON THE HARDWARE IMPLEMENTATION OF PFQ SCHEDULERS

There is a large amount of literature on PFQ algorithms due to their desired properties for QoS support. In this section, we first present the works which provide design approaches to alleviate the implementation challenges presented in Section 5.1.3, we then discuss the previous work on PFQ implementation on hardware platforms.

Searching for the packet with the minimum finish time is a significant component of the overall complexity of any PFQ algorithm. To this end, [19, 96, 99] propose low complexity search hardware designs where [36] proposes the quantization of packet sizes and flow rates to simplify the implementation.

The RAM-based Searching Engine (RSE) [96] is a multi-level implementation of the calendar queue search data structure [100]. Calendar queue trades off the memory use with speed to achieve O(1) time complexity. This data structure is also employed in the scheduling modules of commercial routers [101]. RSE stores the finish times for head-of-line packets in a different RAM for each level of the hierarchy. The number of levels and the number of memory accesses increase logarithmically with the number of possible items that is searched. Hence, the amount of delay to locate the packet with the minimum finish time increases with the depth of the RAM hierarchy. The overflow problem is solved with a two-zone structure which effectively doubles the required memory size for the implementation. It is stated that the RSE design can be realized on FPGA however, there is no such implementation. [99] features a multi-bit tree to accomplish the search in O(1) time which is 4 cycles under best case. The evaluation of the design via hardware simulation with certain assumptions about the traffic shows that 40 Gbps line rate can be reached when this search architecture is used with a PFQ scheduler. [19] proposes a pipelined heap data structure for fast search operations. However, the insert and delete operations of the heap require expensive bus support to complete in a single clock cycle. In addition, increasing the heap size slows down the clock frequency.

[36] assumes that Internet packet sizes and supported rates only take a small number of different values. Accordingly, the authors propose a new two-level architecture called tiered-service fair queuing (TSFQ) which behaves identical to WF^{2}Q+ under certain constraints. The virtual time computation of TSFQ is carried out in one clock cycle provided that the flow rates and packet sizes are selected from a small set. Despite certain packet sizes are more common in the IP packet

distribution, around %30 of packets are smoothly distributed between 40 Bytes and 1500 Bytes. In addition, a certain rate allocation granularity is required to achieve the efficient bandwidth utilization [102]. Searching for the minimum finish time also takes a small number of clock cycles because of the small number of different rates and corresponding queues. However, the searching time is not necessarily a single clock cycle. The implementation and performance evaluation of TSFQ is carried out in Linux kernel without any measurements in hardware.

[103, 104, 105, 106, 107] propose hardware designs for scheduling algorithms with different target FPGA platforms. However, in all these works the implementation is limited to the VHDL model synthesis without any tests or experiments carried out on hardware.

[103] integrates different components of a WFQ scheduler that are described in a series of publications by the same authors including [99]. WFQ has a more complex implementation and a worse WFI compared to WF^2Q and WF^2Q+ algorithms [23]. The design consists of WFQ virtual time computation circuit, a sorting circuit and a high speed shared buffer memory. The implementation platform is an evaluation board with Altera Stratix 2 FPGA. There are 3 pairs of 8 Kbit dual port memory blocks in the system. Flow identifiers are selected to be 13 bits and acordingly the authors compute the number of flows that can be supported by their system as $2^{13} = 8000$ flows. Similarly the bps rate that can be supported by the proposed design is stated to be 12.8 Gbps limited by the memory access speed. The virtual time function of WFQ has a high complexity that grows linearly with the number of connections hence it is the most significant part of the design. However, the authors only state that the virtual time function is implemented with a table look up without any further details. Furthermore no discussion is provided related to the possible limits on the supported number of flows or on the bit rate due to the implementation of the WFQ virtual time function or the pipelined operation of the searching circuit. The experimental results are provided only for 3 flows and it is not indicated if the experiments are performed on real hardware or via simulation.

[104] aims to implement WF^{2}Q+ algorithm on FPGA. To this end, the virtual time function is modified however the modification leads to a different virtual time function which is not equivalent to that of WF^{2}Q+ as defined in [16, 23]. Hence, although the title of the paper suggests WF^{2}Q+ implementation, the implementation outcome operates differently. The overflow problem is solved by subtracting a common value from all vectors. The implementation platform is designated as Xilinx Virtex II 6000 FPGA. However, the scheduling algorithm's performance is measured with an event driven software simulator without any results collected on hardware.

[105], [106] and [107] propose different QoS scheduling algorithms that are designed for low complexity implementation without any investigation towards the fairness and delay bounds of these algorithms. [105] proposes a specific new PFQ algorithm (Gigabit Fair Scheduling-GFS). In this work flows are aggregated into 64 different bandwidth allocations. It is stated that GFS can support 32K flows. [106] carries out the implementation on Xilinx Virtex-I and Virtex-II. The implementation is pipelined which increases the throughput with the cost of delay. [107] proposes a two level design that is composed of round robin schedulers at the first level and highest-level-first selection at the second level. The authors propose using pipelining in their future work to increase throughput. The target hardware platform is Xilinx Virtex-II Pro.

5.3 IMPLEMENTATION OF WF^{2}Q+ ALGORITHM ON FPGA

We apply our general PFQ design framework to a particular PFQ algorithm to demonstrate its features. To this end, we chose the WF^{2}Q+ scheduler which has the best WFI value and a low time complexity for $V(t)$ [23]. It should be noted that WF^{2}Q+ has the same WFI and the end-to-end delay bounds as the predecessor algorithm WF^{2}Q [15]. However, the complexity of WF^{2}Q is higher. For WF^{2}Q+, $V(t)$ is calculated as

$$V(t + \tau) = \max(V(t) + W(t, t + \tau), \min(S_i(t))) \tag{5-5}$$

where $W(t, t + \tau)$ is the number of bits transmitted in the time interval $(t, t + \tau)$.

Note that the complexity of $V(t)$ comes from searching for the minimum $S_i(t)$. We add a third searching module to the $V(t)$ calculator in the general architecture as presented in Figure 5-1 to search for the minimum valued start time vector among all per-flow queues. As this module works in parallel with the other searching modules it does not add an extra delay to the operation. In WF^{2}Q+, a queue i is eligible if its $S_i(t)$ value is not greater than $V(t)$. When the system is not busy with transmitting a packet and if not all the queues are empty, the algorithm selects the eligible queue with the smallest $F_i(t)$ and sends the head-of-line packet to the output channel.

Our implementation of WF^{2}Q+ supports 16 flows for the demonstration of the architecture. Hence, we use a simple comparator tree to implement the searching module. The comparator tree has a depth of 4 and a search operation takes 4 cycles. This searching module is implemented in packet selection module, overflow control module and $V(t)$ calculator module in Figure 5-1. All these modules add a total delay of 4 clock cycles to the operation as they operate in parallel.

Our WF^{2}Q+ design supports 20 different normalized rates (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 30, 60, 90, 120, and 240) and 50 IP Packet sizes where L_{min} = 40 Bytes and L_{max} = 1536 Bytes. We maintain two separate look-up tables for division. In the first table, normalized rates from 1 to 15 are used. In the second table, all of the 20 rate values are used. The scheduler dynamically switches between these two tables.

Using the architecture that is presented in Figure 5-1, the WF^{2}Q+ scheduler is designed on Xilinx ISE design environment for FPGA implementation using VHDL language. The design is implemented on one of the data processing boards in our hardware testbed. In our design, the system has 16 FIFOs for queuing packets from 16 flows. Only the packet headers are stored in the FIFOs. Packet payloads are stored in an external RAM. This enables more efficient use of the FPGA logic resources. Each FIFO is able to store 1024 packet headers of 32 bits. When a packet is selected for transmission, the packet transmission module transmits 16 bits in each clock using Xilinx RocketIO transceiver [93]. The scheduler design operates at a clock frequency of 125 MHz. Hence, it is possible to schedule traffic at 2 Gbps interface speed.

Here we would like to note that it is possible to support a larger number of flows by implementing the FIFO per flow queues on external memory instead of on FPGA. In such implementation there will not be a significant decrease of speed provided that the external memory and the FPGA are implemented on the same card.

We have one more remark about the 4 clock cycles delay of the searching circuit. At 125 MHz clock rate, 4 clock cycles translate into a delay of 32 ns. At 2 Gbps line rate and an average packet size of 265 Bytes, this leads to a small probability of missing the service for a packet. A number of conditions must hold together for a packet to miss the service: First, there must be empty queues in the scheduler when the packet is received. Second, the packet must be delivered to one of these queues within a period of 32 ns before the end of the transmission of another packet. Third, this packet must have the smallest finish number among all the HOL packets. The probability of the occurance of these three conditions all together depends on the system state. However, we can say that it is a really low probability which does not affect the fairness of the scheduler drastically.

The information about the utilization of the FPGA resources for the WF^{2}Q+ scheduler design is presented in Table 5-2.

Table 5-2. The utilization of the FPGA resources for the WF^{2}Q+ scheduler.

Resource	Used	Available	Utilization (%)
Slices	5152	9280	55
4-Input LUTs	9595	18560	51
Flip flops	2793	18560	15
Block RAMs	48	88	54
External IOBs	334	564	59

5.4 PERFORMANCE MEASUREMENT

In this section we present our experimental evaluation of the WF^{2}Q+scheduler that we implement according to the design presented in the previous section. We test our implementation under Poisson traffic with exponentially distributed packet sizes and Markovian on-off bursty traffic with fixed packet sizes generated by the FPGEN hardware traffic generator that we developed and implemented on our network testbed platform [21]. The parameters of the generated traffic can be configured as desired. We measure the average and maximum packet delays in the WF^{2}Q+ scheduler and compare the maximum measured delay for each experiment with the analytical delay bound that is defined below.

Let f_j be a traffic flow with a maximum burst size of σ_j bits and average rate of ρ_j bps that is transmitted through a WF^{2}Q+ scheduler. The allocated rate for f_j is r_j where $r_j > \rho_j$. Let C and L_{max} denote the outgoing line rate in bps and the maximum packet size in the network in bits respectively. Then, the delay bound for a given packet P of flow f_j with L_{max} bits in a WF^{2}Q+ scheduler is [23]:

$$\frac{\sigma_j}{r_j} + \frac{L_{\max}}{r_j} + \frac{L_{\max}}{C} \tag{5-6}$$

In this expression σ_j / r_j and L_{max} / r_j represent the time to serve the maximum size burst of σ_j bits and serving P with a length of L_{max} bits in a GPS-like fluid system at the guaranteed rate of r_j. The traffic is served packet by packet in reality. Hence, the last term L_{max} / C represents the delay due to a packet P' that is from some other flow with a larger finish time than P. P' also has L_{max} bits and starts to get service just before P becomes head-of-line. P has to wait for P' to get transmitted as there is no preemption.

In our experimental set-up we measure the packet queuing delay as the time between the time the first bit of the packet enters the scheduler and leaves the scheduler excluding the packet transmission time on the interface. We perform three experiments with Poisson traffic and exponentially distributed packet times. We choose 50 different packet sizes with a minimum packet size of 40 Bytes and maximum packet size of 1536 Bytes. The mean packet size is 265 Bytes. We ran the experiments with 16 flows which are allocated normalized rates of 1, 2, or 4 which correspond to 71.5 Mbps, 143 Mbps and 286 Mbps respectively. The total mean traffic generation rates of all 16 flows are 700 Mbps, 1 Gbps and 1.6 Gbps corresponding to link loads of 35%, 50% and 80%. The mean per flow traffic generation rate is equal among all flows. For each experiment 10000 packets are collected. The experiment results are presented in the first three rows of Table 5-3.

Table 5-3. Experiment results of the WF^{2}Q+ scheduler implementation.

Exp *i*	Traffic type	% Mean total load	Mean queuing delay (ns)	Maximum queuing delay (ns)
1	Poisson	35	21	232
2	Poisson	50	23.6	4960
3	Poisson	80	30.8	5112
4	Bursty	50	144.1	9312

The maximum packet transmission delay for our experiments is 6.144 μs for 1536 Byte packets. If we add this maximum transmission time to the maximum queuing delay of 5.112 μs that we measure in our experiment we find a total maximum delay of 11.256 μs. Poisson traffic is very smooth. Hence, for these experiments we did not employ a traffic shaper to limit the maximum burst size. We consider the flows with the maximum allocated rate of 286 Mbps and assume that the maximum burst size is 0 for a conservative computation of the delay bound in (5-6). Accordingly we compute a maximum delay bound of (1536 · 8 bits) / (286 · 10^6 bps) = 43 μs which is much higher than our measured maximum delay.

We investigate the delay of our WF^{2}Q+ scheduler implementation under bursty traffic in our fourth experiment. We configure FPGEN to generate 64 Byte packets with an average burst size of 96 Bytes and a total mean traffic generation rate of 1 Gbps that is equally distributed among 16 flows. The allocated rates for the flows are the same as in Poisson traffic experiments. The bursty traffic generated by FPGEN first goes into a leaky bucket shaper which constraints the maximum burst size to 2560 Bytes and the average rate per flow to 75% of the respective allocated rate. We selected a large maximum burst size for the shaper compared to the average burst size of the generated traffic such that the bursts are not smoothed out. The queuing delay data are collected over 12000 packets. The experiment results are presented in the last row of Table 5-3.

The maximum queuing delay in the WF^{2}Q+ scheduler under bursty traffic is measured to be 9.312 μs. If we add up the maximum packet transmission delay of 6.144 μs we find the total maximum delay is 15.456 μs. If we plug in the maximum burst size into (5-6), we compute a maximum delay bound of (2560 · 8 + 1536 · 8 bits) / (286 · 10^6 bps) = 114.98 μs which is again much higher than our measured maximum delay. We also observe that the average queuing delays are quite smaller than the maximum delays.

5.5 CONCLUSIONS

The implementation of the PFQ schedulers plays an important role in the QoS performance of the packet switched networks. In this chapter we present a general block level architecture that shows the common and algorithm specific blocks used in the design of all PFQ schedulers. Next, we identify the design challenges and present solutions to these problems. We use several methods to improve the scheduler implementation efficiency in terms of both operation performance and resource consumption. We select one of the PFQ schedulers which is WF^2Q+ and implement it in the hardware testbed. The packet statistics show that, the scheduler is able to process packets with an average delay of 30 ns even in a traffic load of 80%. We use packet delay statistics to show that the delay performance of the scheduler is within theoretical limits.

Our proposed architecture and implementation methods which are presented in this chapter are applicable to the broad family of PFQ schedulers. These methods help to reduce the implementation complexity and make better use of the advantages of the PFQ schedulers.

CHAPTER 6

A WINDOW BASED METHOD FOR PROVIDING QOS GUARANTEES UNDER FLOW AGGREGATION

The complexity of the scheduling algorithms increases with the quality of the provided service and the number of flows that are scheduled. A solution to support high number of flows is aggregating them to decrease the implementation complexity. The basic problem in flow aggregation is preserving QoS guarantees of the constituent flows in the aggregate. As a result of the greedy behavior of one of these flows, the others may receive decreased delay and fairness performance.

[20] proved that if the flow aggregation is performed fairly and the packet schedulers have certain properties, the end-to-end delay guarantees are preserved with respect to the case that no flow aggregation is performed. [20] presents two different approaches for the design of fair aggregators. The first one is "the basic fair aggregator" which limits the service rate for the aggregated flow to the sum of the reserved rates of the input flows. The second approach is "the greedy fair aggregator" which relaxes this limit only if all input flows have an arrival rate greater than their reserved rates. It is possible that the arrival rate of the flows to be aggregated exceed the total reserved rate temporarily. In such case even if there is available capacity to serve these flows, it will not be utilized.

In this chapter, we present Window Based Fair Aggregator (WBFA) and analytically show that it is a fair aggregator as defined in [20]. Hence, the individual delay bounds of the constituent flows aggregated by WBFA are preserved. Our approach allows the constituent flows to use the full capacity of the output channel until the difference in the service received by the flows reaches a limit. As a result of the increase in the utilization, the average end-to-end delays provided by WBFA are expected to be lower than the basic and greedy fair aggregators proposed by [20]. While increasing the utilization, WBFA also preserves the fairness of service to the aggregated flows. In addition to these, WBFA has low implementation complexity and can be efficiently implemented on hardware.

The remainder of the chapter is organized as follows. In Section 6.1, we summarize and discuss the previous work in the literature on flow aggregation. We present the Window Based Flow Aggregation method in Section 6.2. In Section 6.3, we present an analysis of WBFA and its properties. The hardware implementation and test results are given in Section 6.4. Finally, Section 6.5 gives our conclusions.

6.1 REVIEW OF THE PREVIOUS WORK ON FLOW AGGREGATION

The impact of flow aggregation on end-to-end QoS support is investigated in a number of works in the literature. We develop WBFA based on [20] which shows analytically that if all the schedulers in a network are start time schedulers and all the aggregators are fair, the delay bound is preserved.

[108] shows why network calculus cannot be used in analyzing flow aggregation. This work presents the reasons why network calculus is not successful in the case of flow aggregation. In order to reach a successful performance bound, an optimization problem is constructed. This optimization problem is solved for sink-tree networks and a mathematical expression is derived for the delay bound. It is

shown by numerical experiments that this expression is more successful than that is derived from the network calculus. These experiments are carried on the DISCO Network Calculator.

[109] evaluates the end to end delay performance of the guaranteed rate schedulers both analytically and also using computer simulations. In this work, it is pointed that in a network where guaranteed rate schedulers is used, end to end delay bounds can still be found in the case of flow aggregation. Moreover, it is noted that these delay bounds are generally more successful than the bounds obtained without flow aggregation. The analytical results are justified with the simulations. It is observed that the delay performance improves as the number of hops where flow aggregation is performed increases. Additionally, it is observed that most of the time the delay performance of non-work conserving schedulers is worse than that of work-conserving schedulers.

In [110], IntServ, DiffServ and existing QoS structures are explained. Flow based studies in DiffServ are summarized. The concepts of Flow Aware Networking (FAN) and Flow State Aware Architecture (FSA) are explained. When exploring the effects of flow aggregation on QoS, it is stated that flow aggregation does not increase the average delay. Flow Aggregate Based Services (FAbS) which is a new QoS architecture is introduced. FAbS depends on FSA. FAbS uses flow aggregation and tries to prevent congestion. This architecture is compared with the other QoS architectures from many aspects.

[111, 112] examine the fairness issues among individual TCP flows in a differentiated services network. The experiments show that there is significant variation in the performance seen by individual end users when flow aggregation is used. The aggregate containing more flows outperforms the aggregates with fewer flows in terms of achieved throughput. However, the unfairness is more dominant in the aggregate with bigger no of flows.

[113] introduces a single queue Start Time Fair Queuing (SFQ) scheduler. The received packets are ordered in this aggregate queue according to their timestamps instead of the order of arrivals. Simulation results are used to show the benefits of this approach over other single queue schemes such as Random Early Detection (RED) and FIFO. However, no information is provided about the packet delay characteristics.

There are two models which are used in the modeling of the scheduler algorithms: Guaranteed Rate (GR) server model and Latency Rate (LR) server model. [114] proves that when one scheduling algorithm belongs to one of these models, it also belongs to the other model. GR model depends on providing deadline guarantees by using a virtual time function. LR model depends on the service received by a flow in its burst period. In [114], GR and LR models are investigated analytically and the relation between these models is studied. Also, the constant values in these models are calculated for different scheduling algorithms.

In [115], a scheduling algorithm which uses flow aggregation to improve the number of supported flows without violating the delay requirements is explained. Also, a stateless signaling protocol to be used in this scheduling protocol is presented. A computer simulation is used to compare the connection loss ratio that is observed when this scheduling protocol is used and when it is not used.

After the introduction of the concept of fair flow aggregators in [20], more research is carried to enhance the performance of the fair aggregators. The independency of the per-hop delay on the flow rates and the efficient utilization of the output channel are two important properties that lead to several approaches in the flow aggregation. In some schedulers the per-hop delay is inversely related to the reserved data rate of the flow. This is called rate-dependent delay. In order to achieve a lower delay, the flow is required to reserve a higher data rate. There are also schedulers in which the per-hop delay does not depend on the reserved rate of the flow. This approach is called rate-independent delay. Rate-independent delay is a favorable property in scheduling protocols. Another property that is preferred in a scheduling algorithm is work-conservation. If a scheduler does not leave the output channel idle while it has packets to transmit, the scheduling algorithm is called work-conserving. In such a scheduler, when there are some flows that do not use their share of the bandwidth, some other flows may temporarily exceed their reserved rates and take a greater share on the bandwidth. A good fair aggregator is expected to provide both rate-independent delay and work-conservation. Although the fair aggregation presented in [20] provides rate-independent delay, it fails to provide work-conservation.

With the motivation of building a work-conserving fair aggregator, [116], [117] and [118] propose different aggregation algorithms. While these aggregators succeed in providing work-conservation, they fail to provide rate-independent delay. [117] explains how work conserving flow aggregation can be performed such that the end to end delay bound of each flow is independent of the burstiness of the other flows. Coordinated Aggregation with Isolation (CAI) method is introduced. The performance of this method is analyzed analytically and a mathematical expression is obtained for end to end delay bound. The CAI method suffers from unfairness. [118] enhances this method by introducing the reuse of the deadlines and tries to solve the unfairness problem. However, deadline reuse depends on some conditions. These conditions not only limit the provided fairness but also increase the implementation complexity of the aggregator design significantly.

[119] proposes a scheduling algorithm that is both work-conserving and able to provide rate-independent delay. However, in order to achieve these conditions, the choice of the packet size and the data rates is restricted. In this approach, another problem is that, when some inactive flows become active again, a constituent flow which exceeded its reserved rate temporarily may be denied service for some time.

[120] also presents a work-conserving flow aggregation method that provides rate-independent per-hop delay. In this method, each aggregator assigns a tag to each input packet. The tag is equal to the virtual finish time of the packet at the

aggregator. After aggregation, in each scheduler the packets of the aggregate flow are sorted according to this tag value. The main drawback of this method is that, it requires clock synchronization in the core routers throughout the network. Also, all packets of an aggregate flow and its constituent flows are assumed to have fixed size. When the constituent flows send smaller packets, tag values are still computed according to the fixed packet size hence, the constituent flows cannot use their full rate.

6.2 WINDOW BASED FAIR AGGREGATOR

6.2.1 Preliminaries

In [20], a computer network is viewed as a collection of schedulers and aggregators. An aggregator is defined as a scheduler that receives as inputs a set of flows and produces as output a single aggregate flow by merging the packets of the input flows. The rate of an aggregated flow is taken as the sum of the rates of its constituent flows. In [20], the concept of fair aggregators is introduced. First the perquisites for being a fair aggregator are given. Afterwards, it is shown analytically that when fair aggregators are used, delay bounds can be preserved in spite of flow aggregation. In the context of this work, for each flow *f* and each scheduler *s* along the path of *f*, the following notation is adopted:

R.f	forwarding rate reserved for *f*.
p.f.i	i^{th} packet of flow *f*.
S.s.f.i	start time of packet *p.f.i* from scheduler *s*.
E.s.f.i	exit time of packet *p.f.i* from scheduler *s*.
C.s	capacity of the output channel of scheduler *s*.

The start time of a packet is defined as follows: Assume *s* were to forward the packets of *f* at exactly the rate *R.f*, as if *f* were its only input flow and *C.s* were equal to *R.f*. Then, *S.s.f.i* is the time at which *p.f.i* is forwarded to the output channel of *s*.

A scheduler *s* is said to be a start time scheduler iff for every input flow *f* of *s* and every *i*, $i \geq 1$,

$$E.s.f.i \leq S.s.f.i + \delta.s.f.i \tag{6-1}$$

for some constant $\delta.s.f.i$.

Let *s* and *t* be two consecutive start time schedulers along the path of flow *f*. In [121], it is shown that for every *i*, $i \geq 1$,

$$S.t.f.i \leq S.s.f.i + \Delta.s.f.i \tag{6-2}$$

where $\Delta.s.f.i = \max\{\delta.s.f.x \mid 1 \leq x \leq i\}$.

Using (6-1) and (6-2), for a sequence of start time schedulers $t_1, \ldots, t_k$ traversed by flow *f*, the end to end delay bound in a network consisting of start time schedulers can be written for every *i*, $i \geq 1$, as

$$\begin{aligned} S.t_k.f.i &\leq S.t_1.f.i + \sum_{x=1}^{k-1} \Delta.t_x.f.i \\ E.t_k.f.i &\leq S.t_k.f.i + \delta.t_k.f.i \end{aligned} \tag{6-3}$$

An aggregator can be considered as a scheduler which accepts packets from input flows and transmits these packets on the output channel. The only difference is that in the output channel of the aggregator, as a result of the aggregation, all the packets belong to a single flow. Figure 6-1 shows an aggregator *n* and a scheduler *s* next to *n*.

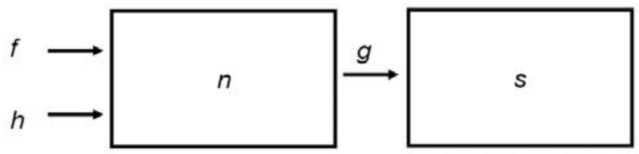


Figure 6-1 An aggregator *n* and a scheduler *s* next to *n*.

An aggregator is fair, if and only if, for every input flow *f* which is aggregated at aggregator *n* with other flows to form the output flow *g*, and for all *i*, $i \geq 1$, there exists a constant $\lambda.n.f.i$ such that

$$S.s.g.j \leq S.n.f.i + \lambda.n.f.i \qquad (6\text{-}4)$$

where *s* is the next scheduler of *g*, and $p.g.j = p.f.i$.

[20] shows analytically that if all the schedulers in a network are start time schedulers and all the aggregators are fair; the delay bound is preserved with respect to the case in which there is no flow aggregation.

It is said that flow *r* is the root of flow *f* at some point in the network if *f* is a constituent of *r* and *r* is not a constituent of any other flow. Let *r* be the root flow of *f* at scheduler *s* and $p.r.j = p.f.i$ for some *i* and *j*, $i \geq 1, j \geq 1$. Then,

$$rp.(s.f.i) = p.r.j \qquad (6\text{-}5)$$

where $rp.(s.f.i)$ is called the root packet corresponding to $p.f.i$ at *s*.

The end to end delay bound for the networks which consist of fair aggregators and start time schedulers is given in [20] by the following theorem.

Theorem: Let *f* be an input flow of scheduler t_1 and let *f* traverse schedulers t_1, ..., t_k. Each of these schedulers is a start time scheduler and is fair. Then, the worst case delay can be written as:

$$\begin{aligned} &S.rp(t_k.f.i) \le S.t_1.f.i + \sum_{i=1}^{i-1} \Lambda.rp.(t_x.f.i) \\ &E.t_k.f.i \le S.rp.(t_k.f.i) + \delta.rp.(t_k.f.i) \end{aligned} \tag{6-6}$$

where Λ is defined as:

If *s* is a fair aggregator with input flow *f*, then for all *i*, $i \ge 1$,

$$\Lambda.s.f.i = \max\{\lambda.s.f.k \mid 1 \le k < i\} \tag{6-7}$$

If *s* is a start time scheduler with input flow *f*, then for all *i*, $i \ge 1$,

$$\Lambda.s.f.i = \Delta.s.f.i \tag{6-8}$$

By comparing (6-3) and (6-6), it can be seen that the end to end delay bound obtained via flow aggregation is similar to the end to end delay bound obtained without flow aggregation. The difference is that while the per-hop delay of packet *p.f.i* at scheduler *t* is Δ.*t.f.i* without flow aggregation, it is Λ.*rp*(*t.f.i*) with flow aggregation. If *t* is a non-aggregating scheduler, then Λ.*rp*(*t.f.i*) = Δ.*rp*(*t.f.i*). This means, per-hop delay of *f* at *t* is the same as the per-hop delay of its root flow at *t*. This shows that the delay is determined by the root flow.

When rate-proportional deadlines are used, each packet *p.g.j* of input flow *g* would exit scheduler *s* no later than *F.s.g.j* + *α.s* where *α.s* is a constant. The per-hop delay of flow *f* at scheduler *s* without flow aggregation is $L_{max}.f$ / *R.f* + *α.s*, and with flow aggregation is $L_{max}.g$ / *R.g* + *α.s*, where *g* is the root flow of *f* at *s*. If *f* is a constituent of g, then $R.f \leq R.g$. As a consequence, delay with aggregation is smaller than the delay without aggregation provided $L_{max}.f \approx L_{max}.g$.

[20] presents two possible implementations of fair aggregators. The first implementation is "the basic fair aggregator". Let n be an aggregator with an output flow g and channel capacity $C.n$. Assume v is a fictitious start time scheduler whose input flows are the same as that of n and whose output channel capacity is $R.g$. In this method, n forwards each packet to its output channel at exactly the same rate that v would forward it.

The disadvantage of this method is that even if the output channel of the aggregator has infinite capacity, the data rate at the output channel of the aggregator cannot exceed $R.g$ which is equal to the sum of the rates of the input flows. The other method is "the greedy fair aggregator". This aggregator has access to a real time clock and maintains a variable that is a function of the elapsed time and the number of bits forwarded from a flow. A packet is forwarded when the real time clock is larger than this variable or when all the input flows have at least one packet in their queues. In this method, the data rate at the output channel of the aggregator can exceed $R.g$. However, the higher rate may be maintained only as long as all input flows have an arrival rate greater than their reserved rates.

The problem about the utilization of the output channel can be better illustrated on an example. Figure 6-2-a shows the arrival times of the three packets of size L to the input of the aggregator n which is shown in Figure 6-1. All three packets are received from the same input flow f. For this case, both the fair aggregator and the greedy fair aggregator forward the packets to the output channel as shown in Figure 6-2-b. The forwarding times are calculated as:

$$t_1 = t_2 - t_1 = t_3 - t_2 = \frac{L}{R.f + R.h} \qquad (6\text{-}9)$$

Although the output channel of the aggregator *n* may have a much larger capacity, the packets are forwarded with a rate *R.f* + *R.h*.

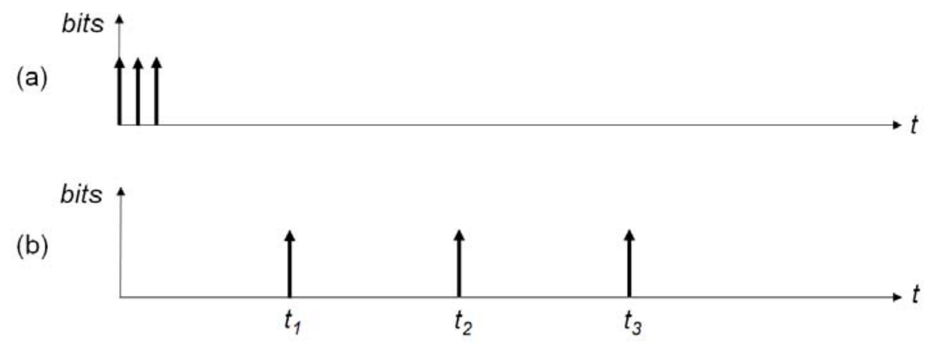


Figure 6-2 (a) The arrival and (b) the departure times of the packets to the aggregator *n*.

6.2.2 Window Based Fair Aggregator

Our proposed Window Based Fair Aggregator (WBFA) is designed to aggregate two input flows into a single output flow such that QoS guarantees can still be provided for the input flows. Although WBFA is designed to aggregate only two flows, it is possible to aggregate larger number of flows by simply cascading several WBFAs back to back.

In the WBFA scheduler, a window based accounting approach is used to provide fairness and delay guarantees for the aggregated flows. In this approach, a window function $w(t)$ is used to keep track of the relative difference in the received service for each flow. The input flows are allowed to use the full capacity of the output channel as long as $w(t)$ is within the predefined limits. The increase in the utilization will help WBFA achieve lower average end-to-end delays than the basic and greedy fair aggregators proposed by [20]. Allowing $w(t)$ to take values within a restricted range also helps to preserve the fairness of service to the aggregated flows. As a result of the easily computable $w(t)$, WBFA has low implementation complexity and can be efficiently implemented on hardware.

6.2.3 Calculation of *w*(*t*)

Initially, when no packets have been transmitted in the flows *f* and *h*, *w*(*t*) is equal to zero. As time progresses, *w*(*t*) is increased and decreased when flow *f* and *h* get service respectively normalized by their reserved rates. We limit the drift of *w*(*t*) from 0 within an amount of w_{max} to preserve the fairness between the services received by *f* and *h*. w_{max} is a parameter to be selected by the designer.

Accordingly, if at time *t*, a packet of flow *f* gets service, *w*(*t*) is updated as

$$w(t) = w(t^-) + \frac{S_f(t,t^-)}{R.f} \tag{6-10}$$

where $S_f(t_1,t_2)$ is the service received by flow *f* between t_1 and t_2. Whenever a packet of flow *h* gets service, *w*(*t*) is updated as

$$w(t) = w(t^-) - \frac{S_h(t,t^-)}{R.h} \tag{6-11}$$

The input flows *f* and *h* have a total reserved rate which is equal to *R.f* + *R.h*. As a result of this, as long as *w*(*t*) is different than 0, *w*(*t*) approaches 0 with a slope *R.f* + *R.h*. Thus,

$$\begin{aligned} w(t_2) &= w(t_1) + (t_2 - t_1)(R.f + R.h) && \text{when } w(t) < 0 \text{ for } \forall\, t,\ t_2 > t > t_1 \\ w(t_2) &= w(t_1) - (t_2 - t_1)(R.f + R.h) && \text{when } w(t) > 0 \text{ for } \forall\, t,\ t_2 > t > t_1 \end{aligned} \tag{6-12}$$

6.2.4 Packet Transmission

The window function $w(t)$ is used to limit the difference in the service offered to the flows f and h. To achieve this, $w(t)$ is allowed to take values between w_{max} and $-w_{max}$. The packets that are received from flows f and h are kept in separate FIFO queues. When a packet becomes head of line in the queue of flow f, the amount of service that is required to transmit the packet is compared with the amount of service that $w(t)$ permits to flow f. Let $P_f(t)$ denote the size of the head of line packet in queue of flow f at time t. If

$$w_{\max} \geq w(t) + \frac{P_f(t)}{R.f} \tag{6-13}$$

then the head of line packet is eligible for transmission and sent to the output channel. Similarly, let $P_h(t)$ denote the size of the head of line packet in queue of flow h at time t. If

$$w(t) - \frac{P_h(t)}{R.h} \geq -w_{\max} \tag{6-14}$$

then $P_h(t)$ is eligible for transmission and sent to the output channel. When the head of line packets of the two flows are eligible for transmission at the same time, one of the packets can be selected arbitrarily. It must be noted that in the bursty intervals of the flows, the window based fair aggregation approach allows the transmission of packets without waiting, as long as the window function does not reach the limit. For the example case that is illustrated in Figure 6-2, as opposed to the basic and greedy fair aggregators, WBFA scheduler would be able to transmit all three packets back to back using full capacity of the channel provided that w_{max} is large enough to allow the transmission of three packets of size L.

When the limit is reached the packets of the "greedy" flow must wait for the window to decrease either suddenly by the service received by the other flow or slowly by the slope *R.f* + *R.h*. The operation of the WBFA aggregator can be better explained on an example. For simplicity, in this example we can assume that the output channel of the scheduler *n* which is shown in Figure 6-1 has an infinite capacity. Figure 6-3 shows the amount of service received by each input flow, *w*(*t*) and the packet departure times for an example incoming traffic pattern.

At *t* = 0, *w*(*t*) starts from 0. Until the arrival of p.*f*.4, the condition given in (6-13) is satisfied and the packets are transmitted immediately. In this period, at the instances of packet transmission, *w*(*t*) increases according to (6-10) and at the other times decreases with a slope *R.f* + *R.h* according to (6-12). When *p.f*.4 arrives, *w*(*t*) is too high to satisfy (6-13) hence, the packet waits for being transmitted until *w*(*t*) decreases enough to satisfy (6-13). In a similar manner, *p.f*.5 waits for *w*(*t*) to decrease enough to satisfy (6-13). When *p.h*.1 arrives, it satisfies the condition (6-14) and is transmitted immediately. When *p.h*.1 is transmitted, *w*(*t*) decreases according to (6-11) and as a result of the sudden decrease in *w*(*t*), (6-13) is satisfied and *p.f*.5 is also transmitted immediately. Similar to *p.h*.1, when *p.h*.2 and *p.h*.3 arrive they are transmitted without waiting. After the transmission of *p.h*.3, *w*(*t*) takes a negative value. From this point on, *w*(*t*) increases according to (6-12) until it reaches 0. After this point, *p.f*.6 and *p.f*.7 satisfy (6-13) and are transmitted as soon as they arrive.

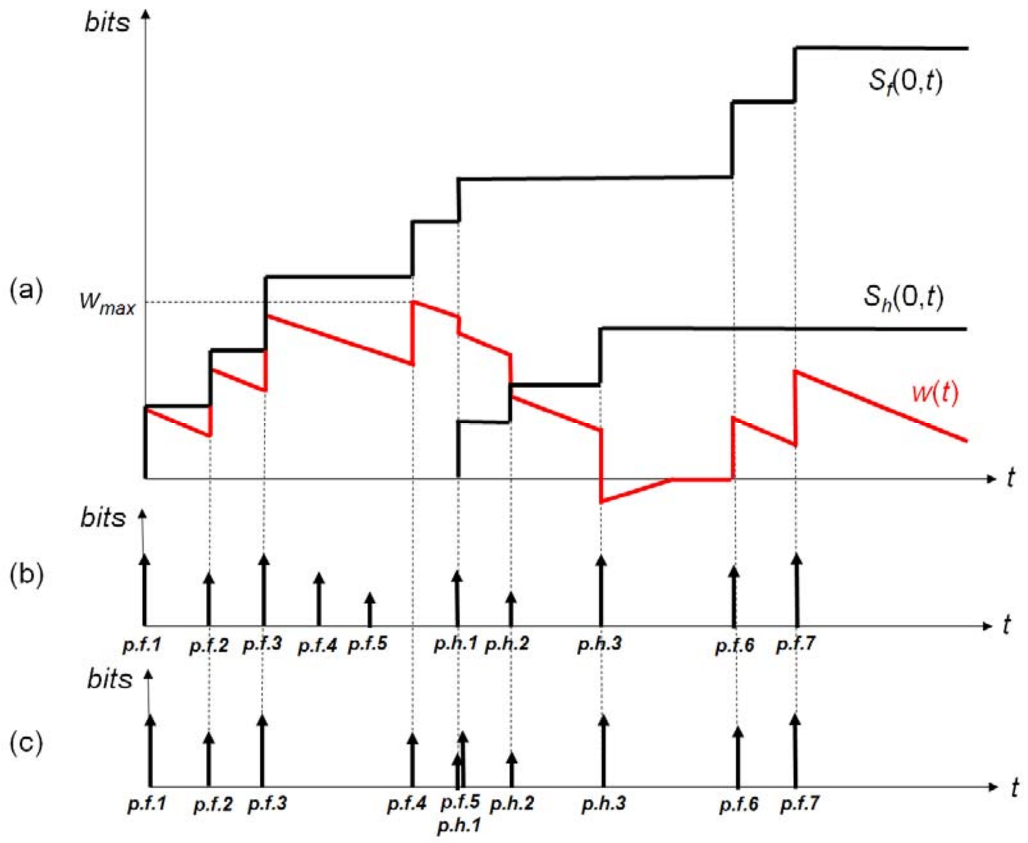


Figure 6-3 An example for the operation of WBFA. (a) shows $w(t)$ and the service received by flows f and h, (b) shows the arrival times and sizes of the packets and (c) shows the departure times of the packets.

6.3 ANALYSIS OF THE WINDOW BASED FAIR AGGREGATOR

In this section, we show analytically that WBFA is a fair aggregator. Let m be an ideal Generalized Processor Sharing (GPS) scheduler based aggregator with an output channel capacity equal to $C.m = R.f + R.h$ and n be a WBFA aggregator with same input flows f, h and output channel capacity $C.n > R.f + R.h$. Let the outputs of the aggregators be forwarded to a start time scheduler s with an output channel capacity equal to $C.s > R.f + R.h$ as shown in Figure 6-4.

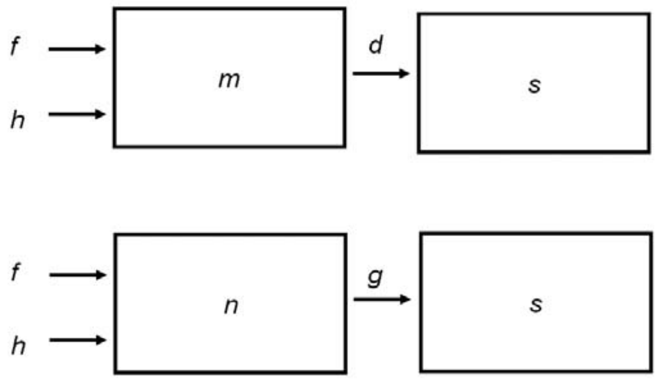


Figure 6-4 The aggregators *m*, *n* and the scheduler *s*.

Lemma 1: The relative ordering of the packets in each of the flows *f* and *h* does not change in the schedulers *m* and *n*.

Proof: In both *m* and *n*, the packets of flows *f* and *h* are kept in separate FIFO queues. Consequently, no packet in these queues can leave before its predecessor.

Lemma 2:

$$w_{\max} \geq \frac{S(0, E.n.f.k)}{R.f} - \frac{S(0, E.m.f.k)}{R.f} \tag{6-15}$$

Proof: The WBFA method allows one of the flows receive service as long as the window function which is updated according to (6-10), (6-11) and (6-12) does not reach the maximum window size.

Lemma3:

$$S.m.f.k = S.n.f.k \tag{6-16}$$

Proof: In Section 6.2.1, the definition of the start time was given as the time that the packet is forwarded to the output channel when each flow is served exactly with its own reserved rate. The flow *f* has the same reserved rate in schedulers *m* and *n*.

Theorem: WBFA aggregator *n* is a fair aggregator.

Proof: As the output channel capacity of *m* is limited to *R.f* + *R.h*, *m* is a basic fair aggregator according to [20] and there exists a constant *λ.m.f.i* such that

$$S.t.d.j \leq S.m.f.i + \lambda.m.f.i \tag{6-17}$$

where $p.d.j = p.f.i.$

Using Lemma 1 and Lemma 2, it is possible to say one can always find a constant *α.m.f.i* such that

$$|k - j| \leq \alpha.m.f.i \tag{6-18}$$

where $p.g.k = p.d.j = p.f.i.$

Then, according to the definition of the start time, one can always find a constant *β.m.f.i* such that

$$|S.s.d.j - S.s.g.k| \leq \beta.m.f.i \tag{6-19}$$

where $p.g.k = p.d.j = p.f.i.$

When we put (6-17) and (6-19) together,

$$S.s.g.k \leq S.m.f.i + \lambda.m.f.i + \beta.m.f.i \tag{6-20}$$

where $p.g.k = p.f.i.$

When we apply Lemma 2 to (6-20), we get

$$S.s.g.k \leq S.n.f.i + \lambda.m.f.i + \beta.m.f.i \tag{6-21}$$

where $p.g.k = p.f.i.$

(6-21) shows that, according to the fair aggregator definition in (6-4), *n* is a fair aggregator and according to the results of [20] it preserves the individual delay bounds of the constituent flows in spite of flow aggregation.

6.3.1 Calculation of the Delay Bound

In (6-6), while $\Delta.s.f.i$ and $\delta.s.f.i$ depend on the types of the start time schedulers used in the network, $\lambda.s.f.i$ is determined by the WBFA aggregator. Hence, for the calculation of the end to end delay bound for a network consisting of start time schedulers and WBFA aggregators, it is necessary to find $\lambda.s.f.i$. Once $\lambda.s.f.i$ is found, the end to end delay bound can be calculated by inserting the characteristic parameters of the start times schedulers which are used in the network into (6-6).

In Figure 6-4, *m* was defined as a GPS scheduler. So, it can be written that

$$E.m.f.i = S.m.f.i + \frac{L.m.f.i}{R.f} \tag{6-22}$$

It was stated that $C.s > R.f + R.h.$ As a result of this, any packet leaving scheduler *m* will start to recive service immediately. Hence,

$$S.s.d.j = E.m.f.i \tag{6-23}$$

where $p.f.i = p.d.j.$ Combining the equations (6-22) and (6-23),

$$S.s.d.j = S.m.f.i + \frac{L.m.f.i}{R.f} \tag{6-24}$$

The maximum value of $S.s.d.j$ is obtained when the packet size is maximum.

$$S.s.d.j \leq S.m.f.i + \frac{L_{\max}.f.i}{R.f} \tag{6-25}$$

From (6-17), $\lambda.m.f.i$ can be written as

$$\lambda.m.f.i = \frac{L_{\max}.f}{R.f} \tag{6-26}$$

As a result of the difference in the order of packets in the flows *d* and *g*, the same packets in different flows will have different start times in the scheduler *s*. The order of the packets is determined mainly by the difference in the service received by each flow in the schedulers *m* and *n*. The scheduler *s* serves each flow packet by packet. Hence, the service difference for the flows *d* and *g* can get worse one maximum size packet in the input of the scheduler *s*.

$$\left|S.s.d.j - S.s.g.k\right| \leq \frac{\left|S(0, E.n.f.i) - S(0, E.m.f.i)\right| + L_{\max}.f}{R.f + R.h} \tag{6-27}$$

Using Lemma 2,

$$|S.s.d.j - S.s.g.k| \leq \frac{(w_{\max}.n)(R.f) + L_{\max}.f}{R.f + R.h} \tag{6-28}$$

From (6-19), *β.m.f.i* can be taken as

$$\beta.m.f.i = \frac{(w_{\max}.n)(R.f) + L_{\max}.f}{R.f + R.h} \tag{6-29}$$

Writing (6-26) and (6-29) into (6-21), *λ.n.f.i* can be written as

$$\lambda.n.f.i = \frac{L_{\max}.f}{R.f} + \frac{(w_{\max}.n)(R.f) + L_{\max}.f}{R.f + R.h} \tag{6-30}$$

6.4 IMPLEMENTATION AND TEST RESULTS

The effect of the flow aggregation of WBFA on the packet delays is measured by implementing WBFA on our hardware testbed. WBFA is designed on Xilinx ISE design environment using VHDL language for FPGA implementation. In our experiments, FPGEN traffic generator generates Poisson traffic at adjustable loads. This traffic initially consists of 16 flows. Later, the 16 flows are aggregated into 4 flows through a two level WBFA array. The two level WBFA array is given in Figure 6-5. Here, w_{max} is selected as 2048. With this selection, even the flow with the smallest reserved bandwidth is able to transmit 10 maximum sized packets before w_{max} is reached. The aggregated traffic is then delivered to a W^2FQ+ scheduler which is explained in Chapter 5. A timestamp is inserted into the header of the packets before they enter the scheduler. At the output of the scheduler, the time that a packet leaves the scheduler is compared with the timestamp in the packet header and the queuing delay is calculated. In this way, the packet queuing delay is measured as the time between the time the first bit of the packet enters the scheduler and leaves the scheduler excluding the packet transmission time on the interface.

Similar to the experiments which are presented in Section 5.4, we did not employ a traffic shaper to limit the maximum burst size.

Table 6-1 summarizes the results of the performance measurement of the WF^{2}Q+ scheduler when WBFA is used to aggregate the input flows. It can be seen that the average delay received by the packets increase with the increasing number of traffic load. However, the maximum delay figure does not have the same tendency. It can be seen that when the traffic load is inceased from 50% to 80%, maximum delay decreases.

Table 6-1 The experiment results of the WBFA implementation.

Exp *i*	No of packets	% Mean total load	Mean queuing delay (ns)	Maximum queuing delay (ns)
1	10000	35	20,08	232
2	10000	50	23,28	5816
3	10000	80	25,84	3808

The reason of this decrease is the more fair distribution of the packets among the queues when the load is increased. This can be more clearly explained on an example. In our implementation, the WF^{2}Q+ scheduler has 4 input queues q_1, q_2, q_3 and q_4 which are assigned to the aggregated flows g_1, g_2, g_3 and g_4 as shown in Figure 6-5. As a result of flow aggregation, there is the possibility that at some instance, g_1 may be consisting of flows f_1, f_2, f_3 and f_4 where g_2 may be consisting of f_5 only. In this case, g_1 will receive a scheduling service share proportional to the sum of the reserved rates of f_1, f_2, f_3 and f_4. However, g_2 will receive a much lower service which is proportional to the reserved rate of f_5 only. What makes the case much more dramatic is that, in addition to the greater service share received by g_1, the number of packets waiting to be served in q_1 is four times the number of packets in q_2. This unfairness may result in large delay for the packets of f_5. However, when the traffic load is increased, the possibility of the aggregated flows being more equally occupied also increases. This explains the decrease in maximum delay when the traffic load is inceased from 50% to 80%.

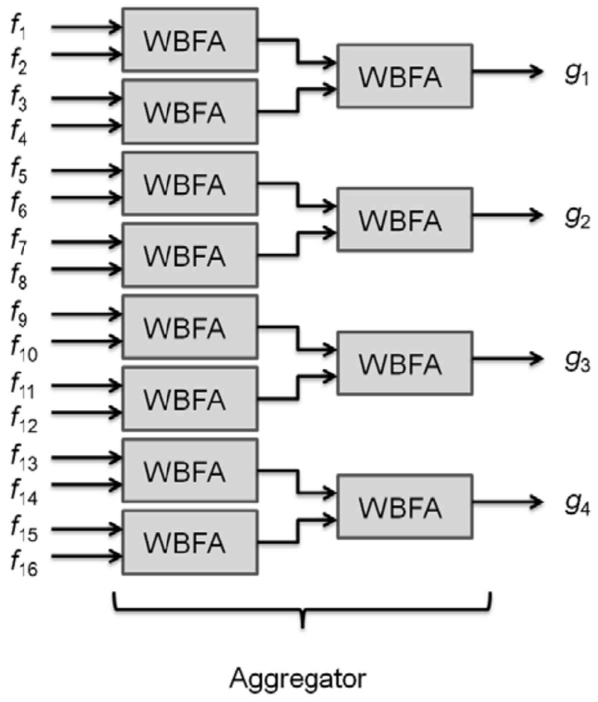


Figure 6-5 The two level WBFA array structure.

In the implementation we used a fair aggregator which is WBFA and a start time scheduler which is WF^{2}Q+. According to [20], when all the aggregators in a system are fair aggregators and all the schedulers are start time schedulers, the flow aggregation preserves, and in some cases improves the delay bound compared to the case where flow aggregation is not used. Table 6-1 tells that the maximum delay obtained in our experiments is 5.816 µs. The maximum transmission delay for our experiments is 6.144 µs for 1536 Byte packets. If we add this maximum transmission time to the maximum queuing delay of 5.816 µs, we get a total maximum delay of 11.96 µs. In Section 5.4 we calculated that, the theoretical delay bound for the WF^{2}Q+ scheduler is above 43 µs for the flow with the tightest delay bound. Hence, the maximum delay in our test results is within the theoretical delay bound even for the flow with the tightest delay bound. We can conclude that using the fair aggregator WBFA and the start time scheduler WF^{2}Q+, the system succeeded in remaining within the theoretical delay bound.

When we compare the average delay figures in Table 6-1 with those of Table 5-3, we see that when WBFA is added to the WF^{2}Q+ scheduler, the average delay is improved according to the case where WF^{2}Q+ scheduler is used on its own.

6.5 CONCLUSIONS

Fair aggregators are used for providing QoS guarantees when flow aggregation is performed in the network. We propose Window Based Fair Aggregator (WBFA) and show that it is a fair aggregator which preserves the individual delay bounds of the constituent flows in the aggregate.

WBFA uses a window function to keep track of the relative difference in the received service for each flow. This method allows the input flows to use the full capacity of the output channel as long as the value of the window function is within the allowed range. The benefit of WBFA becomes more significant if the aggregated flows have similar traffic characteristics such as burst times and allocated rates. In this case the service difference will mostly stay in the allowed window range while the available resources are utilized. We would like to note that [20] also states this conclusion for the proposed aggregators. As a result of the increase in the utilization the average end-to-end delays provided by WBFA would be lower than the basic and greedy fair aggregators proposed by [20].

While increasing the utilization, WBFA preserves the fairness of service to the aggregated flows as we demonstrate in Lemma 2. In WBFA, the service given to each flow differs from the service given in the ideal GPS scheduler by only a constant which can be determined by the designer.

The calculation of the window function does not involve complex arithmetic operations and has low computational complexity. The calculation requires only addition, comparison and division operations. In hardware design, while addition and comparison operations can be implemented efficiently using very little logic resources, division may demand more resources depending on the number of bits involved in the operation. However, it must be noted that the operands *L* and *R.g* which are used in the division operation can take only a finite number of values. Both the number of packet sizes in IP networks and the number of possible reserved rates for the flows has a limited set of values. Hence, by using this set of values, a look-up table can be constructed and division operation can be transformed into simply reading the corresponding entry in a table. In this way, division can also be implemented efficiently by using small amount of logic resources.

The motivation for flow aggregation is to decrease the complexity of the scheduling algorithm. Hence, it is important that the flow aggregation hardware does not increase the overall complexity of the design.

CHAPTER 7

SUMMARY AND CONCLUSIONS

In this thesis, we try to identify the difficulties in the design of PFQ schedulers and overcome these difficulties using different techniques. The first step of our research is the design of a hardware testbed where we can implement schedulers and evaluate their performance on hardware. The hardware testbed consists of three FPGA based data processing boards and a backplane to provide communication and power distribution among the boards. The electrical design, PCB design and manufacturing of all the data processing boards and the backplane are performed in the context of this thesis work. Each of the data processing boards are able to communicate with the other boards through two OC-48 optical fiber channels. These channels provide each board a communication capacity of 5 Gbps.

The next step of our research is the design of a FPGA-based hardware traffic generator that can generate IP traffic to be used in the performance evaluation of the schedulers in the testbed. In today's network environment, design of hardware traffic generators is an independent and hot research problem. The testing and performance evaluation of high-speed network devices requires high-speed packet generators which can generate network traffic at predetermined load conditions and traffic patterns. The software-based traffic generators do not scale to high speeds in the order of Gbps. One other problem with the software traffic generators is that the lack of dedicated hardware resources and CPU-based operation lead to deviation from intended traffic profile at lower speeds than the saturation point of the throughput. Hence, hardware-based traffic generators are required to investigate the performance of the backbone network devices.

Our traffic generator design, FPGEN is developed on FPGA. FPGEN is scalable to high-speeds as it is implemented purely on hardware without using any high level programming or processors. The packet generation times are computed in real-time entirely using the logic resources of the FPGA. FPGEN can generate one packet per clock period, hence it supports up to 125 Mpps per interface at 125 MHz clock rate of our board. Furthermore this rate scales linearly with increased clock rate, number of interfaces or interface speed. Our experiments show that the FPGEN board can support a total traffic generation rate of 5 Gbps and 250 million packets per second with its two OC-48 interfaces. FPGEN is configurable to generate traffic with different parameters due to the programmability of the FPGA. FPGEN generates Poisson traffic with exponentially distributed packet sizes. We present a model which overcomes the inherent difficulties of generating this traffic on a serial interface due to the required independence between the packet sizes and the inter-packet times. In addition, FPGEN can generate Markov-modulated on–off traffic entirely on hardware. We present the design and implementation details of FPGEN followed by an experimental demonstration of achieving packet generation at OC-48 rate per interface. FPGEN serves as the packet generator for the performance evaluation of the developed schedulers.

Next, with the aim of discovering the challenges and difficulties in the design of PFQ schedulers, we construct a system level architecture to identify the functional blocks that are used commonly in all PFQ schedulers and the blocks that are unique to the scheduling algorithms. Using this architecture, we introduce some new design tricks and also make use of some previously suggested approaches to overcome these difficulties. We try to identify the resource-hungry operations and try to find alternative ways of performing these operations. Hardware division is used in many different blocks throughout the design. The scheduler design requires a lot of hardware dividers working in parallel which results in the waste of huge amount of logic resources. We try to prevent the waste of resources by using a look-up-table instead of dividers. In the scheduler design, it is necessary to find the minimum of the time stamps in several different functions. Hence, searching efficiency in terms of speed and logic area affects the performance of the scheduler closely. In our design we use a RAM-based searching engine to enhance the searching performance. Some vectors in the scheduler design are updated continuously. Due to their continuously increasing behavior, after some time these vectors reach their maximum (all bits are ones) and then, overflow and restart counting up from zero. This problem is known as counter aging problem. As a solution to this problem, we apply a counter renewal procedure. In addition to these measures, we use dynamical adaptation to enhance the performance of the scheduler design. The scheduler is given the ability to simplify some calculations according to the set of flows present in the scheduler at the time of operation.

We use our proposed architecture and the design enhancements to implement a popular PFQ algorithm Worst Case Fair Weighted Fair Queuing (WF^2Q+) on our FPGA-based hardware testbed. Using the traffic generated by FPGEN, packet statistics are collected and the performance evaluation of the algorithm is demonstrated. It is seen that the average processing delay received by the packets increase with the increasing traffic load. However, even for a traffic load of 0.8, the scheduler is able to process packets with reasonable delay figures. One important point we note in our design is that, in our PFQ scheduler architecture, the implementation complexity increases with the number of flows that are scheduled. In PFQ schedulers, some of the functions are calculated in per flow basis. The packets of different flows are kept in different queues. Also, there are other operations such as searching and sorting which become more complicated with the increasing number of flows. Hence, for a given amount of logic and hardware resources, when the number of flows exceeds a certain limit, the scheduling algorithm will be unable to continue its proper operation.

A solution to support high number of flows is aggregating them to decrease the implementation complexity. The basic problem in flow aggregation is preserving QoS guarantees of the constituent flows in the aggregate. As a result of the greedy behavior of one of these flows, the others may receive decreased delay and fairness performance. This problem is tackled in [20] which proposes a network model that consists of flow aggregators and packet schedulers. It is proved that if the flow aggregation is performed fairly and the packet schedulers have certain properties, the end-to-end delay guarantees are preserved with respect to the case that no flow aggregation is performed. [20] presents two different approaches for the design of fair aggregators. The first one is “the basic fair aggregator” which limits the service rate for the aggregated flow to the sum of the reserved rates of the input flows. The second approach is “the greedy fair aggregator” which relaxes this limit only if all input flows have an arrival rate greater than their reserved rates. It is possible that the arrival rate of the flows to be aggregated exceed the total reserved rate temporarily. In such case even if there is available capacity to serve these flows, it will not be utilized.

We propose Window Based Fair Aggregator (WBFA) and analytically show that it is a fair aggregator as defined in [20]. Hence, the individual delay bounds of the constituent flows aggregated by WBFA are preserved. Our approach allows the constituent flows to use the full capacity of the output channel until the difference in the service received by the flows reaches a limit. As a result of the increase in the utilization, the average end-to-end delays provided by WBFA are expected to be lower than the basic and greedy fair aggregators proposed by [20]. While increasing the utilization, WBFA also preserves the fairness of service to the aggregated flows. In addition to these, WBFA has low implementation complexity and can be efficiently implemented on hardware.

Following our analytical study, WBFA is implemented in our hardware testbed. In our implementation, the traffic generated by FPGEN is delivered to WBFA for the aggregation of incoming flows to a fewer outgoing flows. Then, the aggregated flows are processed in the WF^{2}Q+ scheduler before reaching the output channel. Our implementation results show that the average delay performance of the overall scheduling system is improved when WBFA is used, compared with the case where there is no flow aggregation.

As a conclusion, the scalability of the scheduler implementations in high speed core networks can be improved by decreasing either the complexity of the per-flow operations or the number of flows in the scheduler. The complexity of the per-flow operations can be decreased by using efficient hardware implementation methods. The scheduling algorithms should take the advantage of being implemented on hardware and make use of the benefits offered by the hardware implementation techniques. The number of flows can be decreased by aggregating the flows. Fair aggregation techniques can be used to preserve the QoS guarantees of the constituent flows when aggregation is used. However, while the non-work-conserving approaches in fair aggregation results in poor output channel utilization, the work-conserving approaches introduce problems about delay performance and implementation complexity. WBFA improves the output channel utilization while still providing tight delay guarantees to the constituent flows. It also helps to provide fairness among the constituent flows. Its low implementation complexity makes it a favorable aggregation method for high speed core networks.

Our entire development and performance evaluation is carried out on FPGA hardware platform. Our future work includes developing a software simulator to conduct experiments with any number of nodes to investigate WBFA QoS support. Furthermore such software simulator can be used to compare WBFA and the basic fair aggregator [20] under similar traffic conditions and the packet delay statistics may be collected to show the increase in average utilization when WBFA is used. In addition to these experiments, some other experiments or simulations may be performed to illustrate the fairness provided to the constituent flows when WBFA is used. In these experiments, per-hop delay statistics may be collected and compared with the delay performance of other fair aggregation approaches.

REFERENCES

[1] J. L. Salina, P. Salina, "Next generation networks: perspectives and potentials," Wiley, 2008.

[2] A. K. Parekh, R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," IEEE/ACM Trans. Networking, vol. 1, pp. 344-357, June 1993.

[3] A. Demers, S. Keshav, S. Shenker, "Analysis and simulation of a fair queueing algorithm," in Proc. ACM SIGCOMM, pp. 1-12, Sept. 1989.

[4] R. L. Cruz, "A calculus for network delay, part I: Network elements in isolation," IEEE Trans. Inform. Theory, vol. 37, pp. 114-131, Jan. 1991.

[5] A. K. Parekh, R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The multiple node case," IEEE/ACM Trans. Networking, vol. 2, pp. 137-150, Apr. 1994.

[6] L. Georgiadis, R. Guerin, V. Peris, R. Rajan, "Efficient support of delay and rate guarantees in an internet," in Proc. ACM SIGCOMM, pp. 106-116, Aug. 1996.

[7] S. J. Golestani, "A self-clocked fair queueing scheme for broadband applications," in Proc. IEEE INFOCOM, pp. 636-646, 1994.

[8] S. Suri, G. Varghese, G. Chandranmenon, “Leap forward virtual clock: A new fair queueing scheme with guaranteed delays and throughput fairness,” in Proc. IEEE INFOCOM, pp. 557-565, 1997.

[9] D. Stiliadis, A. Varma, “Latency-rate servers: A general model for analysis of traffic scheduling algorithms,” in Proc. IEEE INFOCOM, pp. 111–119, 1996.

[10] D. Stiliadis, A. Varma, “Design and analysis of frame-based fair queueing: A new traffic scheduling algorithm for packet-switched networks,” in Proc. ACM SIGMETRICS, pp. 104-115, 1996.

[11] D. Stiliadis, A. Varma, “A general methodology for design efficient traffic scheduling and shaping algorithms,” in Proc. IEEE INFOCOM, pp. 326-335, 1997.

[12] D. Stiliadis, A. Varma, “Rate-proportional server: A design methodology for fair queueing algorithms,” IEEE/ACM Trans. Networking, vol. 6, pp. 164-174, 1998.

[13] D. Stiliadis, A. Varma, “Efficient fair queueing algorithms for packet-switched networks,” IEEE/ACM Trans. Networking, vol. 6, pp. 175–185, 1998.

[14] H. Zhang, S. Keshav, “Comparison of rate based service disciplines,” in Proc. ACM SIGCOMM, pp. 113-122, 1991.

[15] J. C. R. Bennett, H. Zhang, “WF^{2}Q: Worst-case fair weighted fair queueing,” in Proc. IEEE INFOCOM, pp. 120-128, 1996.

[16] J. C. R. Bennett, H. Zhang, “Hierarchical packet fair queueing algorithms,” IEEE/ACM Trans. Networking, vol. 5, pp. 675-689, Oct 1997.

[17] P. Goyal, H. M. Vin, H. Cheng, “Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks,” IEEE/ACM Trans. Networking, vol. 5, pp. 690-704, Oct 1997.

[18] S. Sezer, C. Toal, E. Garcia, V. Stewart, “A reconfigurable tag computation architecture for terabit packet scheduling,” in Proc. 18th International Parallel and Distributed Processing Symposium, 2004.

[19] A. Ioannou, M. G. Katevenis, “Pipelined heap (priority queue) management for advanced scheduling in high-speed networks,” IEEE/ACM Transactions on Networking, vol. 15, no. 2, pp. 450-461, 2007.

[20] J. A. Cobb, “Preserving quality of service guarantees in spite of flow aggregation,” IEEE/ACM Transactions on Networking, vol. 10, no. 1, 2002.

[21] M. Sanlı, E. G. Schmidt and H. C. Güran, “FPGEN: A fast, scalable and programmable traffic generator for the performance evaluation of high-speed computer networks,” Elsevier Performance Evaluation, vol. 68, no. 12, pp. 1276-1290, 2011.

[22] H. J. Chao, X. Guo, “Quality of service control in high-speed networks,” Wiley, 2001.

[23] H. J. Chao, B. Liu, “High performance switches and routers,” Wiley, 2007.

[24] J. Evans, C. Filsfils, “Deploying IP and MPLS QOS for multiservice networks, theory and practice,” Morgan Kaufmann, 2007.

[25] D. Ferrari, D. C. Verma, “A scheme for real-time channel establishment in wide-area networks,” IEEE J. Select. Areas Commun., vol. 8, pp. 368-379, 1990.

[26] H. Zhang, D. Ferrari, "Rate-controlled static-priority queuing," in Proc. INFOCOM, pp. 227-236, 1993.

[27] S. Kweon, K. G. Shin, "Providing deterministic delay guarantees in ATM Networks," IEEE Trans. Networking, vol. 6, pp.838-850, 1998.

[28] S. J. Golestani, "Congestion-free communication in high-speed packet networks," IEEE Trans. Commun, vol. 39, pp.1802-1812, 1991.

[29] C. Kalmanek, H. Kanakia, S. Keshav. "Rate-controlled servers for very high-speed networks," in Proc. GLOBECOM, pp. 300.3.1-300.3.9, 1990.

[30] J. H. Huang, P. C. Tsao, "Continuous framing mechanism for congestion control in broad-band networks," Computer Communications, vol. 18, pp. 718-724, 1995.

[31] I. R. Philip, J. W. S. Liu, "End-to-end scheduling in real-time packet-switched networks," in Proc. International Conference on Network Protocols, pp. 23-30, 1996.

[32] P. Goyal, H. M. Vin, H. Cheng, "Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks," in Proc. ACM SIGCOMM, pp. 157-168, 1996.

[33] S. Cheung, C. Pencea, "BSFQ: Bin sort fair queueing," in Proc. IEEE INFOCOM, pp. 1640-1649, 2002.

[34] S. Ramabhadran, J. Pasquale, "Stratified round robin: A low complexity packet scheduler with bandwidth fairness and bounded delay," in Proc. ACM SIGCOMM, pp. 239-249, 2003.

[35] X. Yuan, Z. Duan, “FRR: A proportional and worst-case fair round robin scheduler,” in Proc. IEEE INFOCOM, pp. 831-842, 2005.

[36] Z. Dwekat, G. N. Rouskas, “A practical fair queuing scheduler: Simplification through quantization,” Computer Networks, vol. 55, no. 10, pp. 2392-2406, 2011.

[37] M. Coss, R. Sharp, “The Network processor decision,” Bell Labs Technical Journal, vol. 9, no. 1, pp. 177-189, 2004.

[38] F. Sabrina, S. Kanhere, S. Jha, “Implementation and Performance Analysis of a Packet Scheduler on a Programmable Network Processor,” In Proc. IEEE Conference on Local Computer Networks, pp. 242-249, 2005.

[39] K. E. Dombkowski, K. F. Kocan, “FPGA technology to minimize extended life-cycle development,” Bell Labs Technical Journal, vol. 9, no. 1, pp. 191-195, 2004.

[40] Miercom, Available online at:
(http://www.miercom.com/?url=services/), last accessed on 1/9/2011.

[41] Spirent, Available online at:
(http://www.spirent.com/Planet-Spirent/SPoC_lab.aspx), last accessed on 1/9/2011.

[42] Ixia, Available online at: (http://www.ixiacom.com/), last accessed on 1/9/2011.

[43] K. Yashigoe, K. J. Christensen, “An evolution to crossbar switches with virtual output queuing and buffered cross points,” IEEE Network, vol. 17, no. 5, pp. 48-56, 2003.

[44] M. Song, W. Zhu, “Throughput analysis for multicast switches with multiple input queues,” IEEE Communications Letters, vol. 8, no. 7, pp. 479-481, 2004.

[45] G. Sapountzis, M. Katevenis, “Benes switching fabrics with O(N)-complexity internal backpressure,” IEEE Communications Magazine, vol. 43, no. 1, pp. 88-94, 2005.

[46] J. Garofalakis, E. Stergiou, “Analytical model for performance evaluation of multilayer multistage interconnection networks servicing unicast and multicast traffic by partial multicast operation,” Performance Evaluation, vol. 67, no. 10, pp. 959-976, 2010.

[47] Z. Zenghao, Y. Yuanyuan, “A novel analytical model for switches with shared buffer,” IEEE/ACM Transactions on Networking, vol. 15, no. 5, pp. 1191-1203, 2007.

[48] N. Chrysos, N. Dimitrakopoulos, “Practical high-throughput crossbar scheduling,” IEEE Micro, vol. 29, no. 4, pp. 22-35, 2009.

[49] D. Zaragoza, C. Belo, “Experimental validation of the on–off packet-level model for IP traffic,” Computer Communications, vol. 30, no. 5, pp. 975-989, 2007.

[50] A. Gupta, V. Sharna, “A unified approach for analyzing persistent, non-persistent and ON–OFF TCP sessions in the Internet,” Performance Evaluation, vol. 63, no. 2, pp. 79-98, 2006.

[51] H. Ferng, J. Chang, “Connection-wise end-to-end performance analysis of queueing networks with MMPP inputs,” Performance Evaluation, vol. 43, no. 1, pp. 39-62, 2001.

[52] I. D. Moscholios, M. D. Logothetis, G. K. Kokkinakis, "Call-burst blocking of ON–OFF traffic sources with retrials under the complete sharing policy," Performance Evaluation, vol. 59, no. 4, pp. 279-312, 2005.

[53] K. V. Vishwanath, A. Vahdat, "Swing: realistic and responsive network traffic generation," IEEE/ACM Transactions on Networking, vol. 17, no. 3, pp. 712-725, 2009.

[54] S. Avvalone, D. Emma, A. Pescape, G. Ventre, "High performance Internet traffic generators," The Journal of Supercomputing, vol. 35, no. 1, pp. 5-26, 2006.

[55] S. Avvalone, D. Emma, A. Pescape, G. Ventre, "Performance evaluation of an open distributed platform for realistic traffic generation," Performance Evaluation, vol. 60, no. 1–4, pp. 359-392, 2005.

[56] R. Simmonds, B.W. Unger, "Towards scalable network emulation," Computer Communications, vol. 26, no. 3, pp. 264-277, 2003.

[57] Traffic Generator, Available online at: (http://www.postel.org/tg/tg.htm), last accessed on 1/9/2011.

[58] Multi-Generator MGEN, Available online at: (http://cs.itd.nrl.navy.mil/work/mgen/index.php), last accessed on 1/9/2011.

[59] RUDE and CRUDE, Available online at: (http://rude.sourceforge.net/), last accessed on 1/9/2011.

[60] D-ITG, Distributed Internet Traffic Generator, Available online at: (http://www.grid.unina.it/software/ITG/), last accessed on 1/9/2011.

[61] A. Botta, A. Dainotti, A. Pescape, “Do you trust your software-based traffic generator,” IEEE Communications Magazine, Computer Communications, vol. 48, no. 9, pp. 158-165, 2010.

[62] J. Jaeger, “FPGA-based prototyping grows up,” Electronic Engineering Times, no. 518, 2008.

[63] B. Kirk, “FPGA-prototyping and ASIC-conversion considerations,” EDN, vol. 52, no. 21, pp. 67-70, 2007.

[64] A. Abdo, H. Awad, S. Paredes, T. J. Hall, “OC-48 configurable IP traffic generator with DWDM capability,” in Proc. Canadian Conference on Electrical and Computer Engineering, pp. 1842-1845, 2006.

[65] G. Salmon, M. Ghobadi, Y. Ganjali, M. Labrecque, J.G. Steffan, “NetFPGA-based precise traffic generation,” in Proc. NetFPGA Developers Workshop’09, 2009.

[66] G. A. Covington, G. Gibb, J. W. Lockwood, N. Mckeown, “A packet generator on the NetFPGA platform,” in Proc. IEEE Field-Programmable Custom Computing Machines, pp. 235-238, 2009.

[67] A. Tagami, T. Hasegawa, K. Nakao, “OC-48c traffic tester for generating and analyzing long-range dependence traffic,” Seventh International Symposium on Computers and Communications, pp. 975-982, 2002.

[68] B. Matthews, I. Elhanany, “A high-speed reconfigurable architecture for heterogeneous multimodal packet traffic generation,” in Proc. IEEE 48th Midwest Symposium on Circuits & Systems, pp. 1143-1146, 2005.

[69] P. Chu, B. Frantz, “A reprogrammable FPGA-based ATM traffic generator,” in Proc. Sixth Great Lakes Symposium on VLSI, pp. 35-38, 1996.

[70] Altera, Available online at: (http://www.altera.com), last accessed on 1/9/2011.

[71] Xilinx, Available online at: (http://www.xilinx.com), last accessed on 1/9/2011.

[72] NetFPGA, Available online at: (http://www.netfpga.org/), last accessed on 1/9/2011.

[73] B. Lemouzy, J. Garnier, N. Neufeld, “FPGA based data-flow injection module at 10 Gbit/s reading data from network exported storage and using standard protocols,” Journal of Instrumentation, vol. 6, no. 2, 2011.

[74] A. A. Alves, “The LHCb detector at the LHC,” Journal of Instrumentation, vol. 3, no. 8, 2008.

[75] M. Frank, J. Garnier, C. Gaspar, G. Liu, N. Neufeld, A. S. Varela, “Online testbench for LHCb high level trigger validation,” Journal of Physics Conference Series, vol. 219, no. 2, 2010.

[76] O. Callot, M. Frank, J. Garnier, C. Gaspar, G. Liu, N. Neufeld, A. S. Varela, A. C. Smith, D. Sonnick, “High-speed data-injection for data-flow verification at LHCb,” IEEE Transactions on Nuclear Science, vol. 57, no. 2, pp. 497-502, 2010.

[77] LHCb ECS, Available online at: (http://cern.ch/lhcb-online/ecs), last accessed on 1/9/2011.

[78] V. Sharma, R. Mazumdar, “Estimating traffic parameters in queueing systems with local information,” Performance Evaluation, vol. 32, no. 3, pp. 217-230, 1998.

[79] F. Baccelli, S. Machiraju, D. Veitch, J. Bolot, “The role of PASTA in network measurement,” IEEE/ACM Transactions Network, vol. 17, no. 4, pp. 1340-1353, 2009.

[80] G. Terdik, T. Gyires, “Lévy flights and fractal modeling of Internet traffic,” IEEE/ACM Transactions on Networking, vol. 17, no. 1, pp. 120-129, 2009.

[81] T. Karagiannis, M. Molle, M. Faloutsos, A. Broido, “A nonstationary Poisson view of Internet traffic,” in Proc. INFOCOM, pp. 1558-1569, 2004.

[82] J. Cao, W. S. Cleveland, D. Lin, D.X. Sun, “Internet traffic tends toward Poisson and independent as the load increases,” Nonlinear Estimation and Classification, pp. 83-109, 2002.

[83] E. Pallares-Segarra, J. Garcia-Haro, “Fluid-flow approach to evaluate the information loss probability in a finite buffering switching node under heterogeneous ON/OFF input traffic sources,” Performance Evaluation, vol. 51, no. 2, pp. 153-169, 2003.

[84] X. Li, I. Elhanany, “Heterogeneous maximal-throughput bursty traffic model with application to packet switches,” in Proc. IEEE/Sarnoff Symposium on Advances in Wired and Wireless Communication, pp. 158-159, 2005.

[85] A. Adas, “Traffic models in broadband networks,” IEEE Communications Magazine, vol. 35, no. 7, pp. 82-89, 1997.

[86] T. Benson, A. Anand, A. Akella, M. Zhang, “Understanding data center traffic characteristics,” ACM SIGCOMM Computer Communications Review, vol. 40, no. 1, pp. 92-99, 2010.

[87] M. E. Crovella, A. Bestavros, “Self-similarity in World Wide Web traffic: evidence and possible causes,” IEEE/ACM Transactions on Networking, vol. 5, no. 6, pp. 835-846, 1997.

[88] A. T. Andersen, B. F. Nielsen, “A Markovian approach for modeling packet traffic with long-range dependence,” IEEE Journal on Selected Areas in Communications, vol. 16, no. 5, pp. 719-732, 1998.

[89] NIST/SEMATECH e-Handbook of Statistical Methods, Available online at: (http://www.itl.nist.gov/div898/handbook/pmc/section3/pmc331.htm), last accessed on 1/9/2011.

[90] P.J. Burke, “The output of a queuing system,” Operations Research, vol. 4, pp. 699-704, 1956.

[91] Xilinx Virtex-II Pro and Virtex-II Pro X FPGA user guide, Available online at: (www.xilinx.com/support/documentation/user_guides/ug012.pdf), last accessed on 1/9/2011.

[92] Linear Feedback Shift Registers in Virtex Devices, Available online at: (http://www.xilinx.com/support/documentation/application_notes/xapp210.pdf), last accessed on 1/9/2011.

[93] Xilinx Rocketio transceiver user guide, Available online at: (www.xilinx.com/support/documentation/user_guides/ug024.pdf), last accessed on 1/9/2011.

[94] NIST/SEMATECH e-Handbook of Statistical Methods, Available online at: (http://www.itl.nist.gov/div898/handbook/eda/section3/eda35g.htm), last accessed on 1/9/2011.

[95] MATLAB version 7.8.0, The MathWorks Inc., 2009.

[96] H. J. Chao, Y. R. Jenq, X. Guo, C. H. Lam, "Design of packet-fair queuing schedulers using a RAM-based searching engine," IEEE Journal on Selected Areas in Communications, vol. 17, no. 6, pp. 177-189, Jun 1999.

[97] A. Lyengar, M. E. Zarki, "Switched prioritized packets," in Proc. IEEE GLOBECOM, pp. 1181-1186, 1989.

[98] Y. R. Jenq, "Design of a fair queueing scheduler for packet switching networks," Ph.D. dissertation, Elect. Eng. Dept., Polytechnic Univ., Brooklyn, NY, 1998.

[99] K. McLaughlin, S. Sezer, H. Blume, X. Yang, F. Kupzog, T. Noll, "A scalable packet sorting circuit for high-speed WFQ packet scheduling," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 16, no. 7, pp. 781-791, 2008.

[100] R. Brown, "Calendar queues: a fast O(1) priority queue implementation for the simulation event set problem," Communications of the ACM, vol. 31, no. 10, pp. 1220-1227, 1988.

[101] K. Dooley, I. Brown, "Cisco IOS Cookbook," O'Reilly Media, Inc., 2006.

[102] R. Bolla, R. Bruschi, F. Davoli, M. Repetto, "Hybrid optimization for QoS control in IP virtual private networks," Computer Networks, vol. 52, no. 3, pp. 563-580, 2008.

[103] K. McLaughlin, D. Burns, C. Toal, C. McKillen, S. Sezer, "Fully hardware based WFQ architecture for high-speed QoS packet scheduling," Integration, the VLSI Journal, (In Press), 10.1016/j.vlsi.2011.01.001, 2011.

[104] M. Song, J. Song, H. Li, “Implementing a high performance scheduling discipline WF2Q+ in FPGA,” in Proc. IEEE Canadian Conference on Electrical and Computer Engineering, pp. 187-190, 2003.

[105] G. Kornaros, T. Orphanoudakis, I. Papaefstathiou, “GFS: An efficient implementation of fair scheduling for multigigabit packet networks,” in Proc. IEEE International Conference on Application-Specific Systems, Architectures, and Processors, pp. 389-399, 2003.

[106] R. Krishnamurthy, S. Yalamanchili, K. Schwan, R. West, “Share-streams: A scalable architecture and hardware support for high-speed QoS packet schedulers,” in Proc. 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 115-124, 2004.

[107] A. Merhebi, O. A. Mohamed, “FPGA implementation of a modular and pipelined WF scheduler for high speed OC192 networks,” 15th ACM Great Lakes Symposium on VLSI, pp. 422-425, 2005.

[108] J. B. Schmitt, F. A. Zdarsky, M. Fidler, “Delay bounds under arbitrary multiplexing: When network calculus leaves you in the lurch,” in Proc. INFOCOM, pp. 1669-1677, 2008.

[109] W. Sun, K. G. Shin, “End-to-end delay bounds for traffic aggregates under guaranteed-rate scheduling algorithms,” IEEE/ACM Transactions on Networking, vol. 13, no. 5, pp. 1188-1201, 2005.

[110] J. Joung, J. Song, S. L. Soon, “Flow-based QoS management architectures for the next generation network,” ETRI Journal, vol. 30, no. 2, pp. 238-248, 2008.

[111] V. Laatu, J. Harju, P. Loula, “The impacts of aggregation on the performance of TCP flows in DS networks,” in Proc. International Conference on Networking, pp. 528-531, 2004.

[112] V. Laatu, J. Harju, P. Loula, “Fairness comparisons of per-flow and aggregate marking schemes in DiffServ networks,” in Proc. The 9th Open European Summer School and IFIP Workshop on Next Generation Networks, pp. 82-87, 2003.

[113] A. Eshete, Y. Jiang, “On the flow fairness of aggregate queues,” in Proc. Baltic Congress on Future Internet Communications, pp. 120-127, 2011.

[114] Y. Jiang, "Relationship between guaranteed rate server and latency rate server," in Proc. Global Telecommunications Conference GLOBECOM, pp. 2415-2419, 2002.

[115] J. A. Cobb, “Scalable quality of service across multiple domains,” Computer Communications, vol. 28, no. 18, pp. 1997-2008, 2005.

[116] W. Sun, K. G. Shin, “Coordinated aggregate scheduling for improving end-to-end delay performance,” in Proc. IEEE International Workshop on Quality of Service (IWQoS), pp. 77-86, 2004.

[117] J. A. Cobb, X. Zhe, “Maintaining flow isolation in work-conserving flow aggregation,” in Proc. Global Telecommunications Conference GLOBECOM, pp. 436-441, 2005.

[118] J. Cobb, Z. Xu, “Guaranteed throughput in work-conserving flow aggregation through deadline reuse,” in Proc. IEEE International Conference on Computer Communication and Networks (ICCCN), pp.87-94, 2006.

[119] J. Cobb, “Work conserving fair-aggregation with rate-independent delay,” in Proc. IEEE International Conference on Computer Communication and Networks (ICCCN), pp. 1-6, 2008.

[120] J. Cobb, “Rate-independent delay across state-reduced networks,” in Proc. Local Computer Networks, pp. 577-584, 2009.

[121] N. Figueira, J. Pasquale, “Leave-in-time: A new service discipline for real-time communications in a packet-switching network,” in Proc. ACM SIGCOMM, pp. 207-218, 1995.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Sanlı, Mustafa

Nationality: Turkish (TC)

Date and Place of Birth: 29 March 1980, İzmir

Marital Status: Single

Phone: +90 312 5921468

Email: musanli@gmail.com

EDUCATION

Degree	**Institution**	**Year of Graduation**
MS	METU Electrical and Electronics Eng.	2005
BS	METU Electrical and Electronics Eng.	2002
High School	Selma Yiğitalp High School, İzmir	1998

WORK EXPERIENCE

Year	**Place**	**Enrollment**
2004 – Present	Aselsan AŞ	Engineer
2002-2004	METU EE Department	Research Assistant
2001 August	Vestel	Intern Engineering Student
2000 August	AR Electronics	Intern Engineering Student

FOREIGN LANGUAGES

Fluent English, Intermediate French

PUBLICATIONS

1. M. Sanlı, E. G. Schmidt and H. C. Güran, "FPGEN: A fast, scalable and programmable traffic generator for the performance evaluation of high-speed computer networks," Elsevier Performance Evaluation, vol. 68, no. 12, pp. 1276-1290, 2011.

2. M. Sanlı, E. G. Schmidt, "Yüksek Hızlı Ağlar İçin Zamanlama ve Anahtarlama Mimarilerinin Tasarımı ve Gerçeklenmesi – Design and Implementation of the Scheduling and Switching Architectures for High-Speed Networks," Gömülü Sistemler ve Uygulamaları Sempozyumu, GÖMSİS2008.

HOBBIES

Windsurfing, seamanship, reading