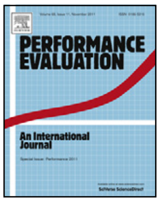
Performance Evaluation 68 (2011) 1276–1290

Contents lists available at SciVerse ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva



FPGEN: A fast, scalable and programmable traffic generator for the performance evaluation of high-speed computer networks

Mustafa Sanlı[a], Ece Güran Schmidt[b,*], Hasan Cengiz Güran[b]

[a] ASELSAN, Ankara, Turkey

[b] Department of Electrical and Electronics Engineering, Middle East Technical University, Ankara, Turkey

ARTICLE INFO

Article history:
Received 24 September 2010
Received in revised form 5 April 2011
Accepted 13 July 2011
Available online 9 August 2011

Keywords:
FPGA
High-speed traffic generator
Poisson traffic
Markov-modulated on–off traffic

ABSTRACT

Testing today's high-speed network equipment requires the generation of network traffic which is similar to the real Internet traffic at Gbps line rates. There are many software-based traffic generators which can generate packets according to different stochastic distributions. However, they are not suitable for high-speed hardware test platforms. This paper describes FPGEN (Fast Packet GENerator), a programmable random traffic generator which is entirely implemented on FPGA (Field Programmable Gate Array). FPGEN can generate variable packet sizes and traffic with Poisson and Markov-modulated on–off statistics at OC-48 rate per interface. Our work that is presented in this paper includes the theoretical design of FPGEN, the hardware design of the FPGA-based traffic generator board (printed circuit board design and construction) and the implementation of FPGEN on FPGA. Our experimental study demonstrates that FPGEN can achieve both the desired rate and statistical properties for the generated traffic.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

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 [1–3].

There is a large number of academic studies on the design and evaluation of different switch architectures, fabric and Quality of Service (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 [4–13]. 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 [14–21] can produce a wide variety of traffic patterns, they cannot reach high packet and bit rates [22]. Hence, design and implementation of high-speed packet generators that can generate the intended traffic properties on hardware is an important research issue.

[*] Corresponding author. Tel.: +90 312 210 4405; fax: +90 312 210 2304.
E-mail addresses: msanli@aselsan.com.tr (M. Sanlı), eguran@metu.edu.tr (Ece G. Schmidt), hasan-guran@metu.edu.tr (Hasan C. Güran).

0166-5316/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.peva.2011.07.003

The commonly used hardware platform for packet generator design in the previous academic literature is *FPGA (Field Programmable Gate Array)*. 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 (Application Specific Integrated Circuit) projects [23]. 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 [24].

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 [25] and external computers [26], or relying on previously collected packet generation statistics at the expense of a memory access for each packet generation [27,28]. Some of these previous studies are implemented and tested on hardware [25–28] while some of them are only simulated in software [29,30]. Furthermore there is no evaluation of the generated traffic according to the desired statistics.

In this paper, 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 paper 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 fiber-optic 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 paper 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 [29,30,25]. However, the used techniques in the previous work are RAM-based and processor based. The third contribution of our paper 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. [25–28] 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 inter-packet 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 our paper is organized as follows. In Section 2, 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 3. 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. We summarize the features of FPGEN in Section 5, after demonstrating them by our experimental studies. Our conclusions are given in Section 6.

2. 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.

Table 1
The evaluation of the previous work on the software traffic generators [22].

Ref., name	Traffic type(s)	Kpps	Mbps
[18], TG	Constant, uniform, exponential, on/off	70	600
[19], MGEN	Constant, exponential, on/off	70	600
[20], RUDE/CRUDE	Constant	80	500
[21], D-ITG	Constant, uniform, exponential, Pareto, Cauchy, normal, Poisson, gamma, on/off	130	500

2.1. Proprietary hardware traffic generators and software hardware traffic generators

Traffic generators can be software or hardware-based. The hardware-based packet generators such as [2,3] 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. [22] present a detailed recent study on the performance of software traffic generators. In this study four packet level traffic generators [18–21] ([21] is also described in [15,16]) are selected according to their popularity.

The results of [22] are summarized in Table 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.

2.2. 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 [23,24].

We present a comparative evaluation of FPGA-based hardware traffic generators in the literature with FPGEN in Table 2. Related work dates back to 1996 motivated by the then high-speed ATM technology [30]. 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.

In [28], 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.

Table 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.
[30], 1996	Markov Modulated Bernoulli Process (MMBP), fixed size (ATM Cell, 53 bytes)	Altera MAX Plus II^2 [31], state machine, sim	12 MHz, 4.5 Gbps
[28], 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
[29], 2005	Bernoulli, 2-state Markov modulated, packet sizes are not specified	Xilinx Virtex-4 FPGA [32], state machine, sim	20 MHz, BW is not specified
[25], 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
[27], 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
[26], 2009	Any traffic type, only packet headers	Xilinx Virtex II Pro 50 FPGA, controls the timing of packets received from a software traffic generator, HW	33 MHz (PCI freq.), total: 1 Gbps (Limited by PCI bus)
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

The arrivals are generated using probability values that are stored in RAMs in [29]. 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.

[25] 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.

[27,26] 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 [33]. 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 [27] or transmitting the packets generated by computer on the Gbps interface [26].

In [27] 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) [26] 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 [27] will scale to generate traffic at these rates. PTG presented in [26] is limited by the PCI bandwidth and will not be able to utilize the high speed interfaces.

2.3. 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 2.1.

Poisson traffic was the first analytical model and is widely studied due to its elegant analytical properties. [4,5] 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. [4,6–9] 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 [34]. This method is widely accepted and recent papers study the cases where using Poisson probes is appropriate [35].

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 [36]. [37] 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 [38], 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 [39,40]. One of the early studies that discuss on–off traffic models is the highly cited work of [41]. These models are verified for contemporary traffic profiles by [10,11] 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. [12,13] study the call and burst level behavior of different service-classes of traffic flows with on–off-bursty traffic. A very recent study [42] shows that the traffic in data centers exhibit on–off behavior.

Lastly, self-similar traffic is another popular and widely studied traffic model [43]. [44] 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.

3. FPGEN Poisson traffic generation

3.1. 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 $\lambda \cdot t$. 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 $\lambda = n \cdot p$. [45] 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.

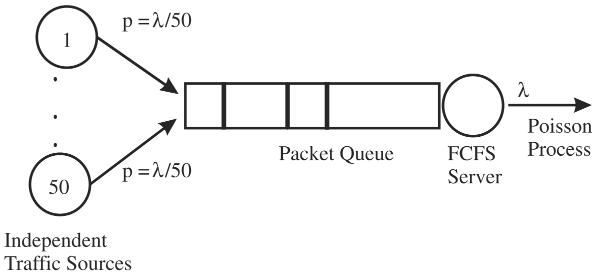


Fig. 1. The design idea of the Poisson traffic generator.

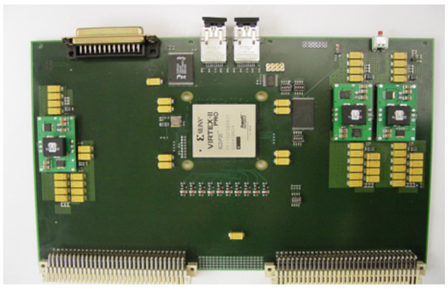


Fig. 2. The traffic generator board.

In our hardware design, we make use of Burke's Theorem [46] 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 Fig. 1.

3.2. Hardware design

FPGEN is designed in the scope of our research for constructing a custom built high-speed network testbed. To this end, we designed a hardware implementation platform which consists of several FPGA based boards that are connected through a backplane. The FPGEN board is part of this platform. However, it is possible to implement FPGEN on any board which contains enough FPGA resources and optical interfaces.

The FPGEN board contains a Virtex2Pro20FF1152 FPGA [47] for the generation, processing and scheduling of packets, a flash memory to store non-volatile data, an EPROM to keep the FPGA code, an RS-232 interface to connect the board to a computer, DC power converters to produce the different voltage levels required by the FPGA, VME-64 connectors for backplane connection and a printed circuit board (PCB) designed with special care to prevent signal coupling between the lines at high data rates. There are two fiber-optic transceivers at OC-48 rate. A RocketIO Multi-Gigabit Transceiver with Aurora core is used to transmit 16 bits in one clock period at the fiber-optic interface with small overhead. At full utilization, 16 bits data can be sent in each clock period at 125 MHz.

The PCB gerber files are designed on Mentor Graphics Design Architect. The PCB has 14 layers and a total thickness of 1.8 mm. The gerber files are simulated with HyperLynx software to detect any signal coupling. The FPGEN board is shown in Fig. 2.

Fig. 3 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.

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.

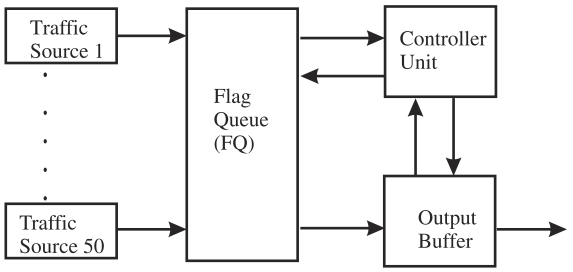


Fig. 3. The basic building blocks of the traffic generator design.

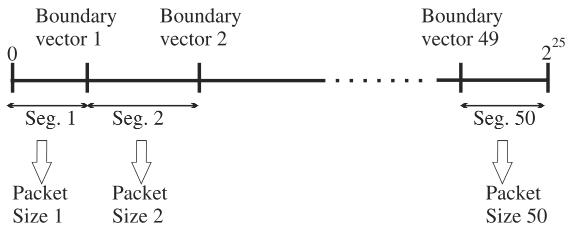


Fig. 4. The boundary vectors and the segments.

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 [48]. In this work, 64 bit LFSRs are used. The LFSRs have 2^{64} 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.

$$4680 \text{ years} = \frac{2^{64} \text{ states}}{125\, M \text{ states/second} \cdot 365 \cdot 24 \cdot 60 \cdot 60 \text{ second/year}}$$

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 [49]. 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. Fig. 4 shows 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 Fig. 5.

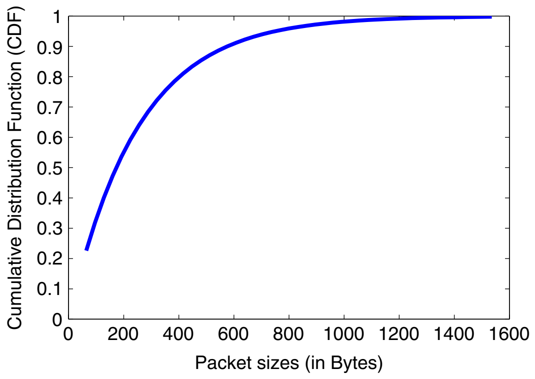


Fig. 5. The cumulative distribution of the packet sizes.

Table 3
The utilization of the FPGA resources for the Poisson packet generator.

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

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 3.

3.3. *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 3.2. In all our experiments, there are 50 different packet sizes which are exponentially distributed between 64 and 1536 bytes as shown in Fig. 5. 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. 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
Experiment loads and data rates. The pps and bps rates are per interface.

Exp. i	ρ_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

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

Exp. i	ρ_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

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 [50].

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

$$F_i(x_{i,j}) = 1 - \exp(\lambda_i^s \cdot x_{i,j})$$

for the exponential distributed inter-packet times. The respective F_i^e and F_i for each experiment i are plotted in Fig. 6.

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

$$d_{i,j} = \frac{\left|F_i(x_{i,j}) - F_i^e(x_{i,j})\right|}{F_i(x_{i,j})} \cdot 100$$

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 [51] 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 5.

Our experiment results show that for low ρ_i values ($\rho_1 = 0.07$, $\rho_2 = 0.15$, $\rho_3 = 0.30$) 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 inter-packet 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 $\rho_4 = 0.60$. 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 $\rho_4 = 0.60$.

The instances where the packet generation time is delayed due to the unfinished packet transmission start to dominate when the load becomes high ($\rho_5 = 0.80$, $\rho_6 = 1$). The inter-packet times start to be mostly determined by the packet

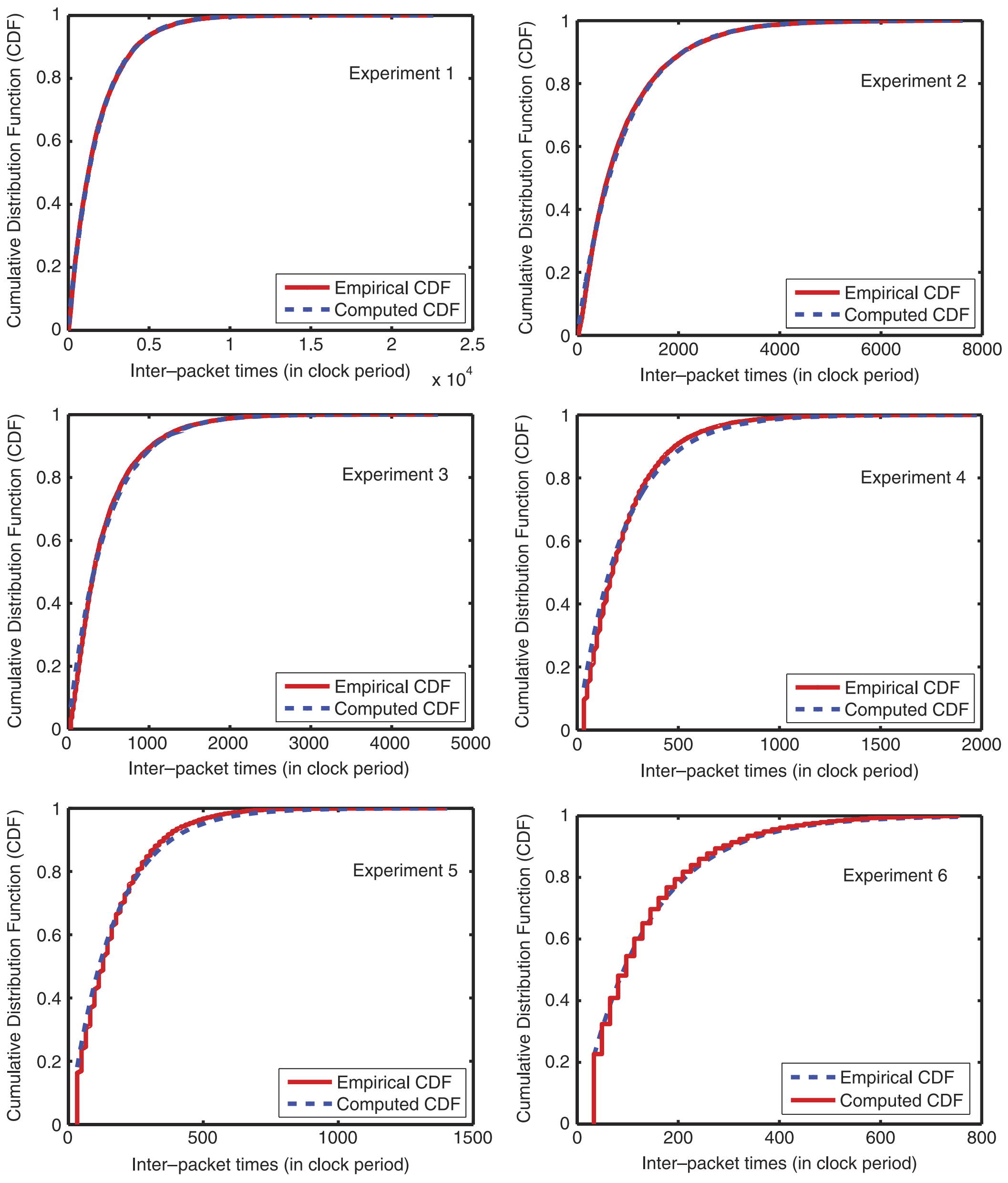


Fig. 6. Experiment results for the comparison of generated traffic distribution and the theoretical expectation.

durations for these load ranges and the inter-packet time distribution starts to get closer to the distribution of the packet sizes that is shown in Fig. 5. 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. $\rho_6 = 1$ 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 Fig. 7. As the empirical CDF has a discrete form the traffic generation with $\rho_5 = 0.80$ passes only at the lowest significance level while $\rho_6 = 1$ fails completely due to the very discrete nature of the sample data.

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 [21,16] which are capable of generating Poisson traffic.

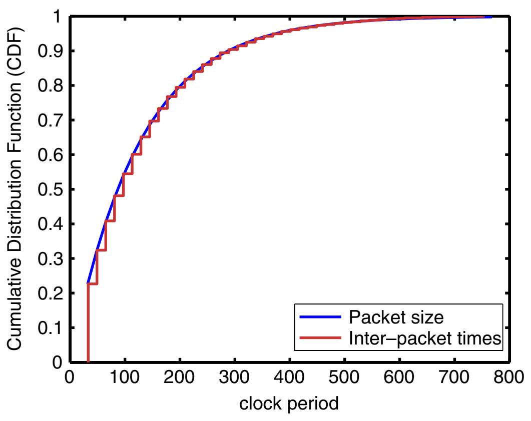


Fig. 7. CDFs for packet sizes and inter-packet times at $\rho_6 = 1$.

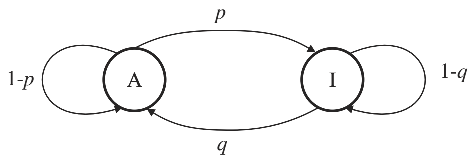


Fig. 8. The state transition diagram of the bursty traffic generator design and the probabilities corresponding to the state transitions.

4. FPGEN bursty traffic generation

4.1. 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 Fig. 8. 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.

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

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

which leads to a mean burst length of

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

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:

$$\rho = \frac{1/p}{1/p + 1/q}.$$

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

$$p = \frac{1}{\beta} \quad \text{and} \quad q = \frac{\rho}{\beta(1-\rho)}.$$

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.

Table 6
The utilization of the FPGA resources for the bursty packet 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 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

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 6.

4.2. 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 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.

5. 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 2.3.

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 5 and 7 show that FPGEN can achieve 2.5 Gbps per interface. The traffic statistics presented in Fig. 6 and Table 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 3.2. 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 2.1.

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 Sections 3.2 and 4.1. 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 FPGEN does not depend on files that contain packet information different than [27,28]. 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 [25,27,26], 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.

6. Conclusion

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 paper 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.

Currently we are working on design and implementation of reconfigurable QoS schedulers on the FPGA platform. FPGEN will serve as the packet generator for the performance evaluation of the developed schedulers. The traffic types generated by FPGEN will be extended to include self-similar traffic in the scope of this study.

References

[1] Miercom, Available online at: (http://www.miercom.com/?url=services/).
[2] Spirent, Available online at: (http://www.spirent.com/Planet-Spirent/SPoC_lab.aspx).
[3] Ixia, Available online at: (http://www.ixiacom.com/).
[4] K. Yashigoe, K.J. Christensen, An evolution to crossbar switches with virtual output queuing and buffered cross points, IEEE Network 17 (5) (2003) 48–56.
[5] M. Song, W. Zhu, Throughput analysis for multicast switches with multiple input queues, IEEE Communications Letters 8 (7) (2004) 479–481.
[6] G. Sapountzis, M. Katevenis, Benes switching fabrics with O(N)-complexity internal backpressure, IEEE Communications Magazine 43 (1) (2005) 88–94. 12.
[7] 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 67 (10) (2010) 959–976.
[8] Z. Zenghao, Y. Yuanyuan, A novel analytical model for switches with shared buffer, IEEE/ACM Transactions on Networking 15 (5) (2007) 1191–1203.
[9] N. Chrysos, N. Dimitrakopoulos, Practical high-throughput crossbar scheduling, IEEE Micro 29 (4) (2009) 22–35.
[10] D. Zaragoza, C. Belo, Experimental validation of the ON–OFF packet-level model for IP traffic, Computer Communications 30 (5) (2007) 975–989.
[11] A. Gupta, V. Sharna, A unified approach for analyzing persistent, non-persistent and ON–OFF TCP sessions in the Internet, Performance Evaluation 63 (2) (2006) 79–98.
[12] H. Ferng, J. Chang, Connection-wise end-to-end performance analysis of queueing networks with MMPP inputs, Performance Evaluation 43 (1) (2001) 39–62.
[13] 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 59 (4) (2005) 279–312.
[14] K.V. Vishwanath, A. Vahdat, Swing: realistic and responsive network traffic generation, IEEE/ACM Transactions on Networking 17 (3) (2009) 712–725.
[15] S. Avvalone, D. Emma, A. Pescape, G. Ventre, High performance Internet traffic generators, The Journal of Supercomputing 35 (1) (2006) 5–26.
[16] S. Avvalone, D. Emma, A. Pescape, G. Ventre, Performance evaluation of an open distributed platform for realistic traffic generation, Performance Evaluation 60 (1–4) (2005) 359–392.
[17] R. Simmonds, B.W. Unger, Towards scalable network emulation, Computer Communications 26 (3) (2003) 264–277.
[18] Traffic Generator, Available online at: (http://www.postel.org/tg/tg.htm).
[19] Multi-Generator MGEN, Available online at: (http://cs.itd.nrl.navy.mil/work/mgen/index.php).
[20] RUDE and CRUDE, Available online at: (http://rude.sourceforge.net/).
[21] D-ITG, Distributed Internet Traffic Generator, Available online at: (http://www.grid.unina.it/software/ITG/).
[22] A. Botta, A. Dainotti, A. Pescape, Do you trust your software-based traffic generator? IEEE Communications Magazine, Computer Communications 48 (9) (2010) 158–165.
[23] J. Jaeger, FPGA-based prototyping grows up, Electronic Engineering Times (518) (2008).
[24] B. Kirk, FPGA-prototyping and ASIC-conversion considerations, EDN 52 (21) (2007) 67–70.
[25] 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, 2006.
[26] G. Salmon, M. Ghobadi, Y. Ganjali, M. Labrecque, J.G. Steffan, NetFPGA-based precise traffic generation, in: Proc. NetFPGA Developers Workshop'09, 2009.
[27] 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, 2009.
[28] 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, 2002.
[29] B. Matthews, I. Elhanany, A high-speed reconfigurable architecture for heterogeneous multimodal packet traffic generation, in: Proc. IEEE 48th Midwest Symposium on Circuits & Systems, 2005.
[30] P. Chu, B. Frantz, A reprogrammable FPGA-based ATM traffic generator, in: Proc. Sixth Great Lakes Symposium on VLSI, 1996.
[31] Altera, Available online at: (http://www.altera.com).
[32] Xilinx, Available online at: (http://www.xilinx.com).
[33] NetFPGA, Available online at: (http://www.netfpga.org/).
[34] V. Sharma, R. Mazumdar, Estimating traffic parameters in queueing systems with local information, Performance Evaluation 32 (3) (1998) 217–230.
[35] F. Baccelli, S. Machiraju, D. Veitch, J. Bolot, The role of PASTA in network measurement, IEEE/ACM Transactions Network 17 (4) (2009) 1340–1353.
[36] G. Terdik, T. Gyires, Lévy flights and fractal modeling of Internet traffic, IEEE/ACM Transactions on Networking 17 (1) (2009) 120–129.
[37] T. Karagiannis, M. Molle, M. Faloutsos, A. Broido, A nonstationary Poisson view of Internet traffic, in: INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies, 2004.
[38] 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 (2002).
[39] 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 51 (2) (2003) 153–169.
[40] X. Li, I. Elhanany, Heterogeneous maximal-throughput bursty traffic model with application to packet switches, in: 2005 IEEE/Sarnoff Symposium on Advances in Wired and Wireless Communication, 2005.
[41] A. Adas, Traffic models in broadband networks, Communications Magazine, IEEE 35 (7) (1997) 82–89.
[42] T. Benson, A. Anand, A. Akella, M. Zhang, Understanding data center traffic characteristics, ACM SIGCOMM Computer Communications Review 40 (1) (2010) 92–99.
[43] M.E. Crovella, A. Bestavros, Self-similarity in World Wide Web traffic: evidence and possible causes, IEEE/ACM Transactions on Networking 5 (6) (1997) 835–846.
[44] A.T. Andersen, B.F. Nielsen, A Markovian approach for modeling packet traffic with long-range dependence, IEEE Journal on Selected Areas in Communications 16 (5) (1998) 719–732.
[45] NIST/SEMATECH e-Handbook of Statistical Methods, Available online at: http://www.itl.nist.gov/div898/handbook/pmc/section3/pmc331.htm.
[46] P.J. Burke, The output of a queuing system, Operations Research 4 (1956) 699–704.
[47] Xilinx Virtex-II Pro and Virtex-II Pro X FPGA user guide, Available online at: (www.xilinx.com/support/documentation/user_guides/ug012.pdf).
[48] Linear Feedback Shift Registers in Virtex Devices, Available online at: (http://www.xilinx.com/support/documentation/application_notes/xapp210.pdf).
[49] Xilinx Rocketio transceiver user guide, Available online at: (www.xilinx.com/support/documentation/user_guides/ug024.pdf).
[50] NIST/SEMATECH e-Handbook of Statistical Methods, Available online at: http://www.itl.nist.gov/div898/handbook/eda/section3/eda35g.htm.
[51] MATLAB version 7.8.0. The MathWorks Inc., 2009.

Mustafa Sanlı received his B.S. and M.S. degrees in electrical and electronics engineering from the Middle East Technical University, Ankara, Turkey in 2002 and 2005, respectively. He is currently studying towards a Ph.D. degree in the same university. His current research interests are in the areas of high-speed networking, scheduling for quality of service guarantees and FPGA-based systems.

Ece Güran Schmidt received her B.S. degree in electrical and electronics engineering from the Middle East Technical University, Ankara, Turkey, in 1997 and her M.S. and Ph.D. degrees in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 2001 and 2004, respectively. She is currently a Faculty Member with the Department of Electrical and Electronics Engineering, Middle East Technical University. Her research interests include high-speed networks, networked control systems, and vehicular communication networks.

Hasan Cengiz Güran received his B.S., M.S. and Ph.D. degrees, in electrical and electronics engineering from the Middle East Technical University, Ankara, Turkey. He is currently a Faculty Member with the Department of Electrical and Electronics Engineering, Middle East Technical University. His research interests include hardware design, microprocessor-based systems and fault-tolerant computing.