A High-Speed Reconfigurable Architecture for Heterogeneous Multimodal Packet Traffic Generation

Brad Matthews, *Student Member, IEEE,* Itamar Elhanany, *Senior Member, IEEE**

Abstract— **Traffic modeling plays a key role in the study of packet switching systems, such as Internet routers. As line rates increase towards tens of gigabits per second, the duration of individual packets decreases, rendering real-time traffic generation a fundamental engineering challenge. In evaluation of these systems, it is critical to reproduce traffic conditions that approximate the target environment. Additionally, the ability to generate traffic flows that establish the limitations of a given algorithm or architecture is highly desirable. To address these issues, we propose a reconfigurable high-speed hardware architecture for heterogeneous multimodal packet generation. FPGA results demonstrate the scalability and flexibility of the proposed framework.**

Keywords— **Traffic Engineering, High-Speed Digital Architectures, Packet Switching**

I. Introduction

Packet switching architectures are represented in many different forms with a primary characteristic being the location of the internal buffering used to store packets. The many variants included input-queued (IQ)[1] switches, output-queued (OQ), combined input-output queued (CIOQ)[2] and shared memory solutions [3] to name a few. Evaluation of these architectures focuses on obtaining performance metrics, such as latency, accepted-versus-offered bandwidth, and jitter [4], under diverse traffic scenarios. In addition to obtaining performance metrics, determining architectural limitations is also highly desirable. Simulation to locate such limitations requires the ability to both vary the load offered to the fabric as well as produce heterogeneous and highly aggregated traffic to replicate the thousands of simultaneous connections found in a typical Internet link [5].

Processing requirements to generate such traffic flows eliminate the use of software-based simulation. A hardware-based approach to ATM traffic generation [6] has been proposed, however our desire is to allow per-flow configurability for traffic aggregation. For this reason, we present an efficient method for the real-time generation of highly aggregated packet traffic in hardware, predominantly for the purpose of evaluating packet switching architectures.

Real-time traffic generation in hardware dictates that the rate at which packets are being presented at each input (ingress) port interface be identical to the maximum line rate supported by the target switch fabric. As line rates have increased towards tens of gigabits per second [7], the duration of individual packet times has decreased, further reducing the available time to generate a packet. Concurrently, the number of ports and flows supported has increased as well, adding considerable complexity to the generation process. The combination of these obstacles renders real-time traffic generation a considerable engineering challenge.

*The authors are with the Electrical and Computer Engineering Department at the University of Tennessee (e-mails: bradmatthews@ieee.org, itamar@ieee.org). This work has been partially supported by the Department of Energy research contract DE-FG02-04ER25607.

II. Traffic Model Elements

Typical packet switching platforms comprise of N ports, each potentially carrying k flows. Letting a flow be tightly coupled with a source-destination pair, we can define flow aggregation to be the dynamic mixing of packets originating from different sources. In this paper, we consider an architecture whereby packets in flows are generated according to either a Bernoulli or two-state Markov-modulated arrival process.

The Bernoulli arrival process produces an uncorrelated stream of packets, with p denoting the probability of generating a packet for any given. Moreover, p represents the offered load, for a Bernoulli arrival process, produced by any given flow. In order to support flow aggregation, packets are selected from one of k flows, with a per-flow probability of $\frac{1}{k}$.

While Bernoulli arrivals are independent, the two-state Markov modulated arrival process generates a stream of correlated bursts that are geometrically distributed in length. For the two-state Markov modulated model presented in Figure 1, arrivals occur when the process is in the ON state, while no packets are generated while the process is in the OFF state. The load offered by the two-state Markov modulated process is given by

$$\lambda = \frac{1-q}{2-p-q}, \tag{1}$$

where p and q denote the probability of remaining in the ON and OFF states, respectively. Each packet in an interleaved burst is associated with a single flow and, consequently, contains identical source-destination pairs. Bursts are again selected from one of k flows. However, a new flow is only selected once the previous burst terminates.

For systems implementing first-come-first-serve (FCFS)-based scheduling, we also present an efficient method for computing the packets' departure times. This provides a mechanism to verify correct implementation of the scheduler without requiring the switch architecture to calculate

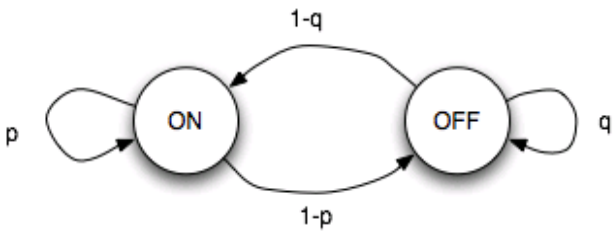

Fig. 1. A two-state Markov-modulated arrival process.

departure times. Moreover, departure times can also be used in performance analysis by conducting a comparison with actual departure times.

III. Traffic Generator Architecture

The traffic generator proposed, as shown in Figure 2, offers a modular architecture that is scalable with respect to the number of packet generation modules and switching port densities. It comprises of five major components: a random number generator, arrival process generator, source-destination lookup, packet formatter and a departure-time calculation module. Each module, with the exception of the departure time calculation module, can be replicated N times and is capable of generating traffic from one of k flows. The scalability property of the departure time calculation module is derives from its $O\log(N)$ complexity that enables N departure times to be calculated, according to a FCFS scheduling algorithm, in a single time slot.

In establishing the arrival process, we have stated support for both the Bernoulli and two-state Markov-modulated arrival processes. In the case of the Bernoulli process, packets are uncorrelated and possess no temporal properties. Thus, each packet can be viewed as having a unique source-destination pair from cycle to cycle. For this reason, the Bernoulli arrival process module is designed to compare a random arrival process value, r_a, produced by the random number generator, with the user-configurable per-flow probability, p_k, in order to determine whether a packet should be transmitted on a per-cycle basis. The value of p_k represents the probability that a packet will be transmitted when flow k is selected. This set of k probabilities is stored in the Source-Destination RAM, with the index denoting the flow selected, using the random flow value, r_f, that is again produced by the random number generator. By dividing the n-bit number space of r_f into k distinct regions, the flow index can be determined using comparators that identify the k^{th} region in which r_f resides. Upon calculation of both r_a and r_f, the packet formatter is instructed to send a packet when the following inequality holds $r_a \leq p_k$.

The two state Markov-modulated arrival process is realized as an extension of the Bernoulli arrival process architecture. The per-flow probability is again selected using the random flow value, r_f, by dividing its n-bit number space into k distinct regions to determine the k^{th} region in which r_f resides. However, the flow index is now used to

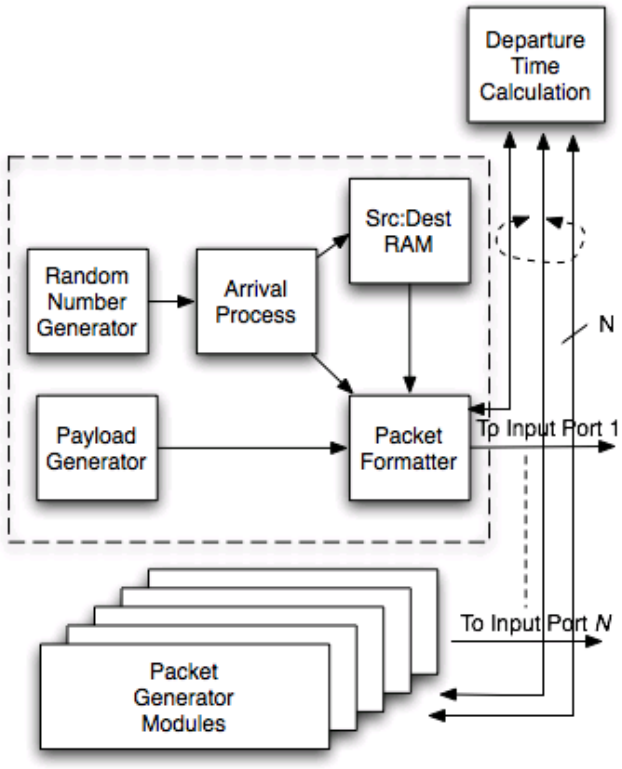

Fig. 2. Traffic generator architecture comprising multiple packet synthesis modules and a departure time calculation unit.

retrieved two flow probabilities, p_k and q_k. The value of p_k represents the probability that the Markov chain remains in the ON state and continues to transmit packets, while the value of q_k represents the probability that the chain remains in the OFF state and does not generate a packet. These transition probabilities are compared to the random arrival process value, r_a, during each cycle to determine whether a state transition should occur. The bursty nature of the two-state Markov-modulated process can be maintained by storing new flow probability values only when the chain transitions to the OFF state.

Prior to transmitting a packet to the packet formatter, the arrival process module must also produce a source-destination pair. This is done as a preliminary step to the packet construction. The source-destination pair is stored in a RAM of depth k. The entire solution can be realized with three RAMs located in the source-destination RAM module; two RAMs contain the arrival probabilities, $p_k$and q_k, while the third RAM contains only the destination address for the packet. Given that the source address is identical for all k flows of any packet generator module, it can be represented by a single register that is programmed during the initialization phase.

Once the target destination for packets presented at each port has been determined, the associated departure time can be calculated prior to transmission to the switch fabric. The packet generation architecture we consider supports systems implementing first-come-first-serve (FCFS)-based scheduling. Packets arriving at a single port can have

monotonically increasing departure times based on their arrival order. To calculate the departure times for a system with N ports, one must be able to compute departure times for the adversarial case where all N packets are destined for the same output port. The computation of departure times in a serial order would require that each computation completes in $\frac{50}{N}$ n sec (for 10 Gbps links). Given that packet switching architectures consisting of 64 ports or more are common, this would require that each departure time be calculated in less than one nanosecond. Clearly, this is not practical. Instead, departure times must be calculated in parallel at the cost of silicon area.

In Figure 3, the architecture of the departure time calculation fabric is depicted. Since departure times monotonically increase as packets arrive at a given output, a register file containing the current departure times for each output is stored in the base departure time register file. To compute the departure time at each port, an encoded destination address presented to the fabric input is first split into an N-bit vectored request, whereby a single active bit represents the packet destination. Once the N-bit vector has been generated for each destination address, the base departure time offset is computed by summing the q^{th} bit, where q identifies each output port in each N-bit vector. The result of the summation represents the number of packets arriving at output port q. This base departure time offset is then added to the base departure time at the end of each clock cycle. Such computation contains the critical path in the design due to the requirement of N $log_2(N)$-to-N multiplexors, whose output is immediately fed to a set of N adders that sum the N bits. This summation is then added to the base departure time register prior to being latched.

Next, we describe the method for calculating per-packet departure times. The departure time for each packet is a function of the destination address for every one of the packets arriving at the same time cycle, and the base departure time for each output port. Since we have the base departure time for each destination, all that is needed is to assign an offset relative to other packets arriving during the same time slot. The offset for each packet is calculated relative to the input port at which it was received. This effectively establishes a priority to each input port in terms of calculating its departure time offset. The first input port automatically has the highest priority. For each successive port, a comparator is required to determine if a higher priority input contains a matching destination address. The result of each boolean comparison is appended to the packets departure time offset. From this, we can establish that a packet arriving at input port 1 is automatically given a departure time of 1. Hence, input port 2 will require one comparator and possess a departure time of either 1 or 2. For input port N, a total of $N-1$ comparators is required to produce a departure time that can vary from 1 to N. Once the departure time offset is calculated, it is added to the base departure time offset register, specified by the destination address, to obtain the per-packet departure time. The selection of the base offset register requires N N-to-1 multiplexors since each packet can be associated with one of the N possible destinations.

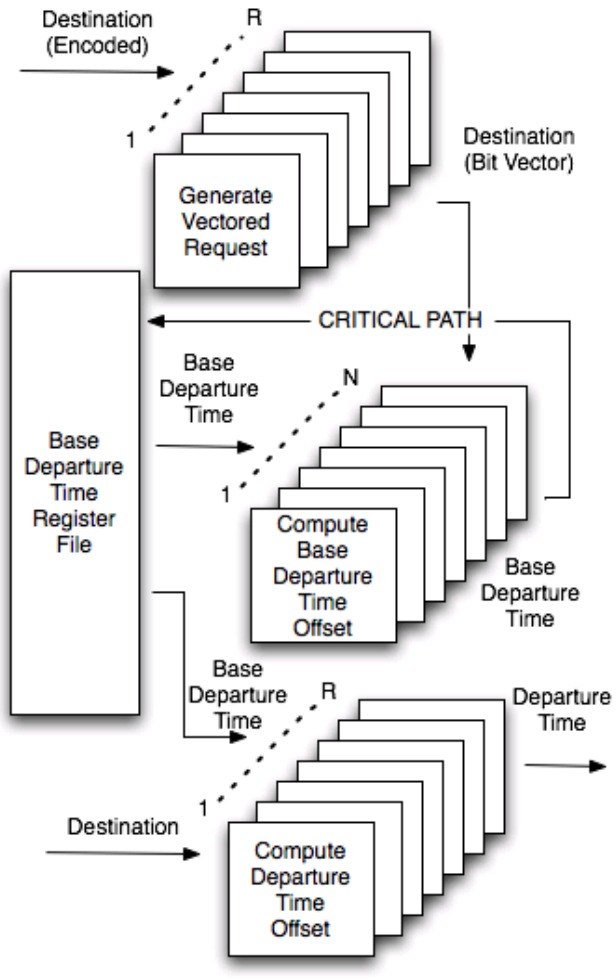

Fig. 3. Departure Time Calculation Architecture

Once the departure time has been calculated, its output is forwarded to the packet formatter which is responsible for synchronizing the departure time, destination and packet transmission information. The departure time and packet validity are considered as verification information. For this reason, such information is embedded in the payload, by the packet formatter, for evaluation and performance analysis purposes at the output of the switch. Upon completing the formatting process, the packet is presented to the ingress interface of the packet switch fabric.

IV. Implementation Results

A. FPGA Realization

The design of the packet generator architecture has been coded in Verilog HDL and synthesized using the Synplicity SynplifyPro synthesis tool, targeting a Xilinx Virtex-4 XC4VLX80-11-FF1148 FPGA device [8]. The implementation consists of 64-ports and supports 128 flows. Packets are produced at the output of the packet generator every *50 ns*.

TABLE I
FPGA Implentation Cost and Expected Slack

Module	LUTs	Slack (ps)
Departure Time Calculation	24674	+0.657
Packet Generator	19214	42.83
Complete Design	43793	+0.27

In order to maximize the utilization of the target devices, we proposed to reduce the logic resources required to compute departure times by limiting the computations to $N/2$, or 32, departure times in a single cycle. By clocking the departure time calculation fabric at least 2 times faster than the rest of the system, 64 departure times could be computed in the required 50 ns period. However, synchronization of the two clock domains necessitates an additional two clock cycles. For this reason, the departure time computation fabric effectively operates at 12.5 ns, or 4 times faster than the rest of the system.

The full implementation of the 64-port packet generator required 43,793 Lookup-Tables (LUTs), or 61% of the LUTs available in the target device. With the goal of 20 MHz operation frequency, synthesis was able to meet timing with *27 ps* of slack Since three RAMs (two for probabilities and one for the source-destination pairs) are required by each generator module, a $3N$ RAM architecture resulted. Accordingly, this implementation consumed 192 of the 200 RAMs made available by the device.

Lookup-Table (LUT) resource consumption, presented in Table 1, reflects the implementation cost of each module. The departure time computation fabric fits comfortably in this device suggesting that the actual computation fabric could be designed to operate at the same clock rate as the rest of the system.

B. Software Simulation

Correct operation of our packet generator architecture was verified using stimulus written in SystemC, and co-simulated with the Verilog HDL implementation, using the Synopsys VCS-MX simulator. Programming the individual packet generators, as well as simultaneously starting all packet generators after configuration, was performed using this stimulus. Configuration of the device was carried out using a single shared bus with a per-generator address decoder that is located at the top-level of the packet generator implementation. The user has the ability to set values for each of the $2k$ flow probabilities, the k source-destination pairs, seeds for the two random number generators, and the arrival process type on an individual packet generator basis. Hence, the duration of the configuration process for the entire system consists of $(2k + 3)N$ clock cycles.

Once the device is configured, each packet generator is enabled to synthesize packets concurrently. To validate the correct operation, the load offered by each packet generator was measured as means of determining the variance between the expected offered load and the offered load produced via simulation. Table 2 illustrates the simulation results for the 64-port packet generator configured to produce a normalized offered load of 0.8. The variance between the expected and offered load decreases as the simulation length increases, indicating that a real-time implementation would yield results that closely approximates the desired offered load.

TABLE II
Offered Load Variance

Simulated Samples	Expected Offered Load	Actual Offered Load
1500	0.8	0.806832
200,000	0.8	0.800771
$1x10^7$	0.8	0.799683

V. Conclusion

In this paper, a reconfigurable high-speed hardware architecture for heterogeneous multimodal packet generation has been presented. The proposed architecture provides flow aggregation using both Bernoulli and two-state Markov modulated arrival processes. Furthermore, FPGA implementation and simulation results were presented to emphasize the viability of this architecture. Future work will focus on supporting additional arrival processes, incorporating quality of service (QoS) provisioning, as well as studying architectural trade-offs to enable increased port density support in FPGA-based realizations.

References

[1] I. Elhanany and D. Sadot, "Disa: A robust scheduling algorithm for scalable crosspoint-based switch fabrics," *IEEE Journal of Selected Areas in Communications*, vol. 21, pp. 535–545, May 2003.

[2] J. Dai and B. Prabhakar, "The throghput of data switches with and without speedup," *IEEE INFOCOM 2000*, pp. 556–564, March 2000.

[3] S. Iyer, R. Zhang, and N. McKeown, "Routers with a single stage of buffering," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 251–264, 2002.

[4] I. Elhanany, D. Chiou, V. Tabatabaee, R. Noro, and A. Poursepanj, "The network processing forum switch fabric benchmark specifications: An overview," *IEEE Network Magazine*, pp. 4–9, March/April 2005.

[5] S. Floyd and V. Paxson, "Difficulties in simulating the internet," *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, 2001.

[6] E. Z. T. Antonakopoulos and V. Makios, "On mapping stochastic processes into hardware and its application on atm traffic emulation," *IEEE Transactions on Instrumentation and Measurement*, vol. 52, no. 3, 2003.

[7] S. Iyer and N. McKeown, "Analysis of the parallel packet switch architecture," *IEEE/ACM Transactions on Networking*, vol. 11, pp. 314–324, April 2003.

[8] Xilinx Virtex-4 technical documentation available at http://www.xilinx.com.