High-performance implementation of in-network traffic pacing — Source link

Y. Sinan Hanay, Abhishek Dwaraki, Tilman Wolf

Institutions: University of Massachusetts Amherst

Published on: 04 Jul 2011 - High Performance Switching and Routing

Topics: TCP pacing, Packet switching, Network packet, NetFPGA and Throughput

High-Performance Implementation of In-Network Traffic Pacing

Y. Sinan Hanay, Abhishek Dwaraki and Tilman Wolf
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA, USA
{hanay,dwaraki,wolf}@ecs.umass.edu

***Abstract*—Optical packet switching networks promise to provide high-speed data communication and serve as the foundation of the future Internet. A key technological problem is the very small size of packet buffers that can be implemented in the optical domain. Existing protocols, for example the transmission control protocol, do not perform well in such small-buffer networks. To address this problem, we have proposed techniques for actively pacing traffic to ensure that traffic bursts are reduced or eliminated and thus do not cause packet losses in routers with small buffers. In this paper, we present the design and prototype of a hardware implementation of a packet pacing system based on the NetFPGA system. Our results show that traffic pacing can be implemented with few hardware resources and without reducing system throughput. Therefore, we believe traffic pacing can be deployed widely to improve the operation of current and future networks.**

***Index Terms*—small-buffer network, traffic pacing, traffic burstiness, field-programmable gate array, prototype**

I. Introduction

High-bandwidth data communication is one of the key foundations for the future Internet. Society's expectations to perform many daily activities over the Internet (e.g., entertainment, person communication, e-government, health care, etc.) has increased the demand for raw bandwidth in the network. One technology that promises to meet these demands for bandwidth is optical networking. Optical fiber has long been used for long-haul, point-to-point data transmissions. However, the need to convert from the optical domain back to the electronic domain for switching has limited the throughput that can be achieved optical-electronic-optical (OEO) networks.

To address these performance limitations, all-optical networks (AON) have been developed. Optical circuit switching (OCS) can be used to provide end-to-end light paths. While OCS has been successfully deployed for some use case scenarios, the slow timescales at which connections can be set up and torn down does not match the timescales of highly dynamic end-to-end Internet connections. As alternative, optical packet switching (OPS) [1] and optical burst switching (OBS) [2], [3] has been proposed. In these networks, packets or short bursts of packets are switched independently in the optical domain.

While OPS and OBS are promising approaches to optical networks that match the needs of current Internet protocols and dynamics, there are important technological challenges that need to be addressed. Apart from the difficulties of building a switch that operates in the optical domain, there are also challenges imposed by the basic operation of the Internet. Packet switching in the Internet is based on statistical multiplexing and traffic is forwarded opportunistically without prior resource reservations. As a result, traffic may compete for link bandwidth on the output port of a switch. To avoid packet losses, packet buffers are used to queue packets and thus absorb short periods of overload. In electronic networks and OEO networks, these buffers can be implemented easily with SRAM or DRAM and can be sized to hold a large number of packets. In all-optical networks, buffers are more difficult to implement since there exist no practical solution to store light other than sending it through a delay line [4]. These optical buffers can only hold a very small number of packets. One important question is how efficiently can such small-buffer networks operate with network, transport and application layer protocols that have been designed for large buffer networks. The issue of buffer sizing has also been explored in other contexts (e.g., to reduce the cost of electronic routers by reducing the amount of buffer memory per port).

One key concern with small-buffer networks is their interaction with the transmission control protocol (TCP) [5]. In particular, the use of congestion control in TCP [6] leads to significant performance drops in throughput when packet loss occurs. Several studies have explored how traffic characteristics (e.g., burstiness) impact packet losses in small-buffer networks and how these losses affect TCP throughput [7]. While there are arguments that aggregation of traffic in the network leads to smoothing of traffic bursts [8], there are also arguments that the conditions for smoothing cannot be ensured [9]. In this context, we have proposed in our prior work to actively smooth traffic in the network and thus achieve the conditions that are necessary for small-buffer networks to operate efficiently [10]. In particular, we have shown that burstiness in traffic can compound in the network and lead to larger average queue lengths in the core (and thus higher packet loss rates for small-buffer networks) [11]. To circumvent this problem, it is possible to perform pacing of traffic and thus reduce or eliminate traffic bursts [12].

In this paper, we present a high-performance implementation of the traffic pacing technique described in our prior work. A key aspect of the design of our traffic pacing method is that

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0721790.

it is easy to implement and can be performed at high data rates. Our prototype system based on the NetFPGA platform [13] shows that traffic pacing can be implemented with a small amount of hardware resources and does not impede the throughput performance of the system. Specifically, the contributions of this paper are:

- A hardware design for a high-performance traffic pacer based on algorithms described in [12], [14],
- A prototype implementation of the traffic paced on the NetFPGA platform, and
- Evaluation results to demonstrate the performance and effectiveness of the hardware design.

We believe that the results from this work are important since they demonstrate that it is possible to implement traffic pacing at high data rates with little additional hardware. Therefore, traffic pacing can be widely deployed and thus help in ensuring that network traffic characteristics are suitable to fully utilize a future Internet that is based on high-performance small-buffer routers.

The remainder of this paper is organized as follows. Section II discusses related work. Section III presents queue-length based pacing, which we have developed in prior work and which is the basis for the hardware implementation presented in this paper. Section IV presents the design of the pacer implementation and a discussion of various design tradeoffs. Results from the prototype implementation and its performance are presented in Section V. Section VI summarizes and concludes this paper.

II. Related Work

The size of router buffers is important for effective operation of networks with statistically multiplexed traffic. The use of the transmission control protocol (TCP) for most data communication in the Internet, has led to buffer sizing recommendations based on the produce of round-trip time and link capacity. Such large buffers ensure that buffer underflows can be avoided when TCP reacts to packet loss (indicating congestion) with multiplicative decrease in transmission rate. However, such buffers are also expensive and difficult to build for optical switches. Recently, there have been several studies showing that aggregation of multiple connections leads to traffic characteristics that ensure efficient operation even with smaller buffer sizes [15]. Appenzeller et al. show that buffer size proportional to square-root of the number of flows may be sufficient [8].

While small-buffer networks can operate efficiently with TCP for some scenarios, it has also been shown that this is not the case for many practical scenarios [9]. In particular, the presence of TCP connections with short lifetimes are problematic. Haesegawa et al. show that a high ratio of short-lived flows has a negative effect on link utilization for small buffers [16]. Sivaraman et al. [17] show the impact of small buffers on real-time and TCP traffic and identify short-timescale burstiness as the major contributor of performance degradation.

Instead of relying on multiplexing effect to smooth out traffic bursts, there have been several approaches to actively change the characteristics of network traffic to meet the needs of small-buffer networks. Pacing for TCP on end-systems has been proposed (albeit not in the context of small-buffer networks) [18]. This approach is difficult to implement since it requires software modifications on all end-systems. In contrast, we have proposed to implement pacing functions on nodes in the network to simplify deployment [10]. A specific pacing algorithm is presented in [12], [14]. A similar technique was developed independently by Alparslan et al. [19]. The hardware implementation of the traffic pacer presented in this paper is based on the algorithms described in these papers.

There have been numerous works analyzing performance of TCP pacing or traffic pacing through ns-2 simulations [14], [16], [18]. To our best knowledge, the only experiment to measure pacing performance is done in [14], using a software-based pacer plugin. In our work, we implement a pacer system in programmable hardware.

III. Operation of QLBP Packet Pacer

As the basis for discussion of the hardware design of the high-performance pacer presented in Section IV, we briefly review the general operation of the traffic pacer described in [10], [12], [14]. The figures, notation, and some text in this section are repeated from these papers to provide the necessary background for the reader.

A. Traffic Pacing in Network

The traffic pacing that we describe in this paper is implemented in electronics and thus cannot be part of a all-optical network. Figure 1 shows our perspective on how we can deploy traffic pacing in a future Internet with an all-optical core. By conditioning traffic in the edge networks, the all-optical core can operate efficiently – even with very small buffers. Pacers can be deployed opportunistically; traffic that traverses multiple pacers achieves better throughput in the small-buffer core, but there is no requirement that a pacer has to be traversed. Also, pacing is done indiscriminately for all traffic; there is no per-flow adjustment of the pacing rate, but all traffic is handled the same.

B. Pacing Algorithm

The pacing mechanism considered in our work is Queue Length Based Pacing (QLBP) [12], [14]. The goal of the pacer is to approximate constant bit-rate (CBR) traffic by delaying certain packets. However, since the pacer cannot know the arrival times of future packets, it is not able to perfectly determine the transmission times that would correspond to CBR. Instead, it uses an adaptive process to determine if and how much packets should be delayed.

The length of the packet queue for an output port is used to determine the transmission rate in QLBP. The transmission rate $\mu(t)$ at time t is determined by queue length $q(t)$ as follows:

$$\mu(t) = \begin{cases} \frac{\mu_{max}-\mu_{min}}{Q_{max}} q(t) + \mu_{min}, & q(t) < Q_{max} \\ \mu_{max}, & q(t) \geq Q_{max} \end{cases} \quad (1)$$

Parameter Q_{max} is the maximum queue length at which pacing is allowed and μ_{min} and μ_{max} are the minimum and

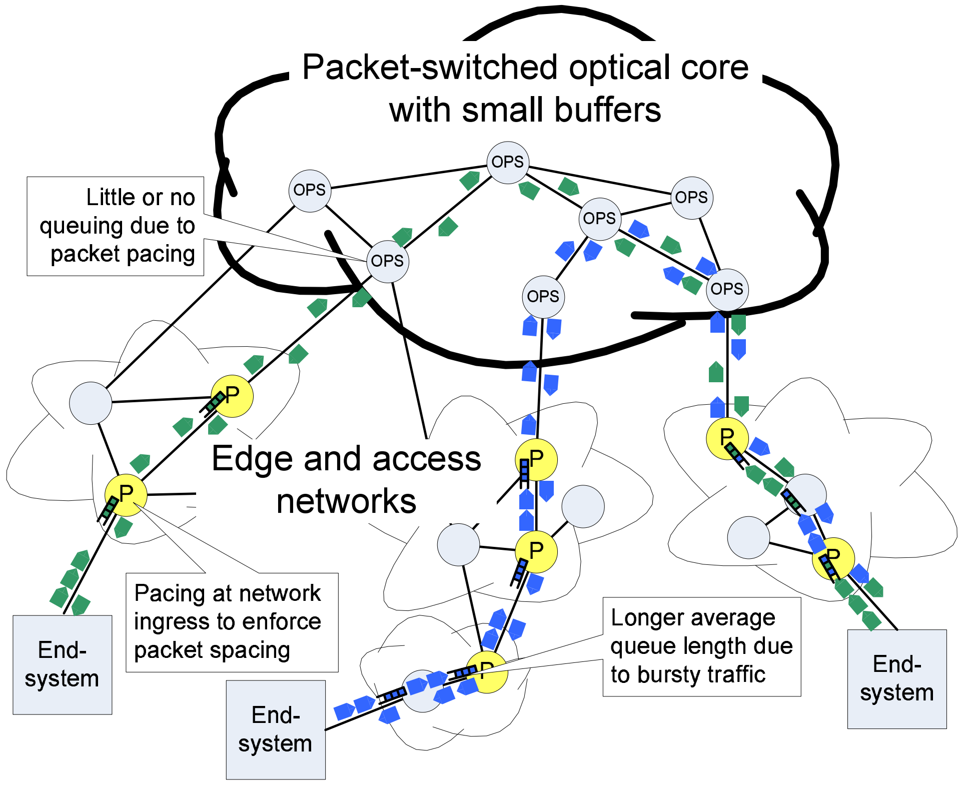
Packet-switched optical core with small buffers
OPS
Little or no queuing due to packet pacing
Edge and access networks
P
Pacing at network ingress to enforce packet spacing
End-system
Longer average queue length due to bursty traffic

Fig. 1. Network Architecture with Traffic Pacing (from [10]).

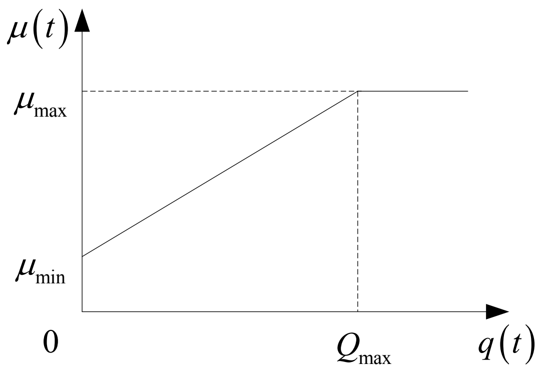

Fig. 2. Pacing Rate of QLBP (from [14]).

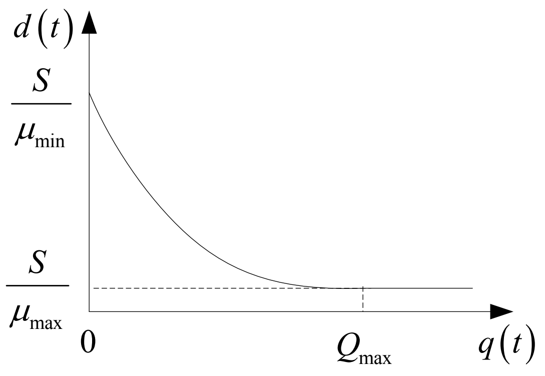

Fig. 3. Pacing Delay of QLBP (from [14]).

maximum transmission rates. Figure 2 shows the transmission rate of the pacer as a function of queue length.

Based on the transmission rate of the pacer, the delay experienced by a packet is

$$d(t) = \frac{S}{\mu(t)}, \tag{2}$$

where S is the size of the packet. Figure 3 shows the delay as a function of queue length. In the implementation of QLBP, the pacing delay is activated after a packet is transmitted to ensure that individual packets that do not belong to a burst do not get delayed.

Thus, QLBP has the following properties that make it effective for use in small-buffer networks:

- For longer queue lengths, the pacing delay is lower. This rule ensures that the link can be fully utilized.
- For an empty queue, the maximum delay is limited. This rule ensures that packets do not get delayed indefinitely.
- The first packet arriving at the pacer does not get delayed. This rule ensures that pacing is only activated when multiple packets (i.e., a burst) are observed.

C. *Effectiveness of QLBP Pacer*

Figure 4 shows the effectiveness of our pacing approach. Traffic bursts traversing the multiple pacing nodes are smoothed out to nearly match constant bit-rate traffic. Using this QLBP pacing at multiple nodes in the network edge ensures that traffic in the core nearly matches CBR traffic and thus can efficiently utilize small-buffer networks.

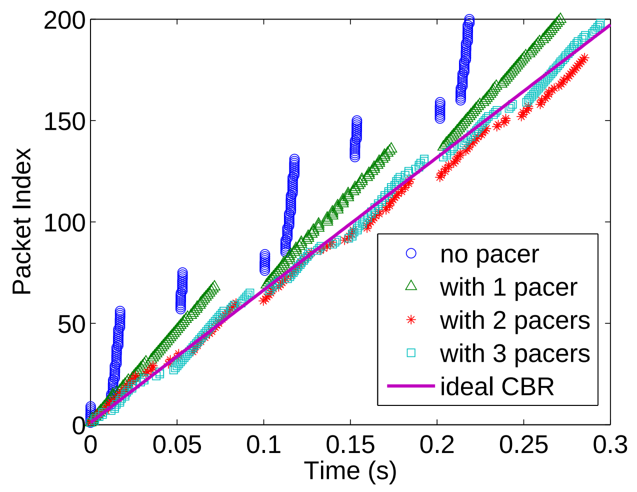

Fig. 4. Effectiveness of Pacing on Multiple Nodes (from [14]).

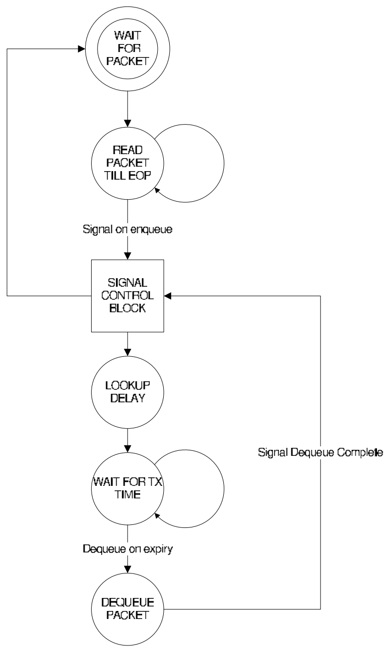

Fig. 5. State Diagram of Pacer Implementation.

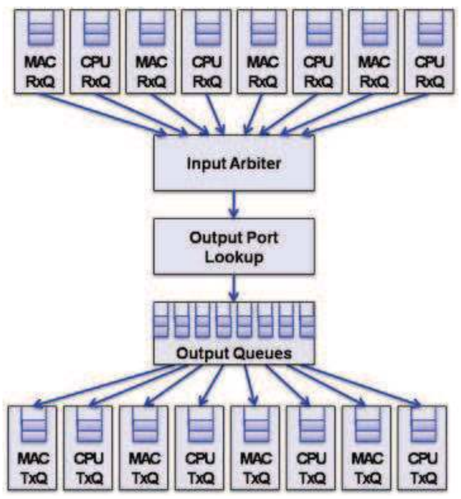

Fig. 6. NetFPGA Reference Router Pipeline (from [20]).

IV. High-Performance Design of Packet Pacer

To ensure that traffic pacing improves the throughput performance of a small-buffer network, it is critical that the traffic pacer itself does not present a performance bottleneck. Therefore, it is important to develop a suitable high-performance design of such a system. We discuss our implementation in this section and evaluation results from the prototype in Section V

This section deals with the architecture of the pacer and how it has been implemented in hardware. We look at the underlying platform that is used and the reference designs that have been modified to suit our needs. We first discuss the NetFPGA in more detail.

1) The NetFPGA Platform: The NetFPGA is a Gigabit Ethernet open platform for networking research which has been developed at Stanford University [13]. It consists of a Xilinx Virtex 2Pro FPGA along with a Spartan 3 FPGA built onto a PCI card along with SRAM, DRAM and 4 Gigabit Ethernet ports. This PCI card plugs into the PCI socket of any PC and can be used to develop and test new networking methodologies and protocols.

2) The NetFPGA IPv4 Reference Router: The IPv4 reference router is part of the NetFPGA base package and consists of library modules connected together in a pipelined fashion to enable the design to operate at almost 125 MHz [20]. It is essential to understand the modular design of the NetFPGA to make optimal use of the existing design. Below is shown the pipeline structure of the NetFPGA reference design. We first take a look at that design and then go into the details of our design. Figure 6 shows the Reference Router pipeline.

There are 4 Gigabit Ethernet ports on the PCI card, which serve as ingress/egress ports for the design. Each of the MAC ports have a corresponding CPU DMA port that acts as the conduit between the NetFPGA card and the host PC via the PCI bus. These DMA ports are used to relay packets to the host PC's processor in the eventuality of the NetFPGA being unable to process them.

Apart from the Rx/Tx ports, the NetFPGA has other modules that are part of the reference design. One of them is the input arbiter, which takes packets from each of the Rx ports in a round robin manner and hands them off to the output port lookup module. Another is the output port lookup module, which is responsible for the IP/LPM lookups and Ethernet header modifications before the packet is pushed to output queues, which places it in the queue for the corresponding output MAC.

The pipelining is done in such a way that each module consists of a small input FIFO where the previous module writes data into. The current module picks up data from here to process. This enables the pipeline to go ahead unhindered. In case a certain module is not ready to accept data, the input FIFO is filled with data and then the pipeline begins to stall backwards until the input buffers are full and packets get dropped.

3) Packet Pacer Design: In this section, we look at the design of the packet pacer on the NetFPGA and how is fits into the reference pipeline. Figure 5 shows the state diagram of the pacer. The packet pacer is been split into various individual modules handle discrete functionalities. Each of the modules is explained below.

- The Input State Machine: This state machine handles the job of monitoring the input FIFO for data. Once data is available in the input FIFO and the pacer module is ready to process data, it transfers the packet from the input FIFO to the pacer FIFO where it is held till it reaches its transmission time. The FIFOs are explained in the next section.
- The FIFOs: There are two FIFOs, which handle the incoming packets. One is 72 bits wide, and holds the packet and some control information. The other is 32 bits wide and holds the size of packets that come into the module. The smaller 32 bit FIFO has a N-1 mapping to the larger FIFO and the size of the packet at the head of the queue is held in a global register, which is explained below.
- The Signal Control Block: The Signal Control Block is responsible for maintaining the current queue size, previous transmission time and next transmission time data. Once the input state machine enqueues a packet, it sends a signal to the signal control block, which first updates the queue size to reflect the new packet and then activates the calculate block to calculate the next transmission time of the packet. The timer control embedded in the control block monitors the validity of the transmission time. When the timer expires on a calculated transmission time value, the control block sends a signal to the output state machine asking it to dequeue the packet at the head of the queue. Once the output state machine has dequeued a packet, the signal control block receives the acknowledgement and updates it current queue size to reflect the new changes.
- The Delay Lookup/Calculation block: This block is responsible for updating the signal control block with the new transmission time on every enqueue or dequeue. Depending on the implementation, it incorporates an array of floating point cores or a Block RAM based Single port ROM. The latency of this block is also dependent on the implementation, with the floating point implementation imparting greater accuracy, but also adding a lot more to the latency due to the complexity involved. On the other hand, the single port ROM does not add much in terms of latency itself since the lookup is very fast, typically one cycle, but approximates the calculated delay using pre-calculated, extrapolated values.
- The Timer Control Block: This block creates a reference time-line for the timer functionality. It consists of a timer that up-counts on every clock and simultaneously monitors whether there next transmission time has been reached. Once the timer hits the next transmission time, it sends a signal to the output state machine to start dequeuing the packet. The block is part of the Signal Control Block.
- The Output State Machine: This state machine is activated by the Signal Control block once the timer has expired and the next transmission time has been reached. It is responsible for handling the packet dequeue and also making sure that the packet size global register is updated with the new packet's size.

The whole pacer module is connected between the output port lookup module and the output queues module on the NetFPGA. This is the initial design and makes it global to the router. All packets entering the router are paced. As a next step to this, the pacer module is replicated and the granularity shifted to per-port pacing.

A. Implementation of the Delay Calculation Block

The critical path in our design is in the delay calculation block. It takes two inputs: packet size, S_p, and instantaneous queue size, $q(t)$, calculates the delay, and outputs t_{next}. t_{next} is calculated as follows:

$$t_{next} = t_{last} + S_p/(\frac{\mu_{max} - \mu_{min}}{Q_{max}} \cdot q(t) + \mu_{min}) \quad (3)$$

In the above equation, the packet size S_p and the instantaneous queue length $q(t)$ are inputs to the calculation black, while u_min and u_max are parameters. We denote the right term in the addition as the, d, delay in our discussion from now onwards, that is:

It can be seen that to calculate delay, d, a straightforward approach requires two divisions, one multiplication and one addition at the minimum, not to mention a host of conversions from integer to the IEEE 754 format to perform the floating point operations.

In the sections below, we discuss alternative hardware implementations of this delay calculation. We present an exact implementation that employs using floating point operations for precision. We also present a look up table based implementation for higher throughput.

1) Floating Point Implementation (FPI): In our implementation, we use floating point blocks to precisely calculate the d. In doing so, first step is to convert the unit of d from time to clock cycles, that is $d_{clk} = f_{clk} \times d$. At this point a designer may want to implement the delay block straightforwardly by using 2 dividers, 1 multiplier and 1 adder. In addition to these, floating to fixed point converters are needed to convert fixed values. Division is a time consuming operation even in hardware. The division $\frac{\mu_{max} - \mu_{min}}{Q_{max}}$ can be transformed to a shift operation if Q_{max} is selected to be a power of 2. Q_{max} can be selected conveniently as a power of 2 for buffer sizes less than 500. For a detailed discussion of Q_{max}, reader can refer to [12]. We have used IP cores provided by Xilinx to perform the floating point operations such as division, addition, multiplication, fixed to float and the vice versa conversions. Floating point divider is pipelined so that it completes in 4 cycles.

2) Lookup Table Implementation (LTI) : It is possible to avoid expensive operations such as divisions and multiplications by storing pre-calculated delay values in a lookup table. The inputs to this LTI block are $q(t)$ and S_p, and look-up table can be implemented in Block RAM as a single-port ROM. The valid range for Q_{max} is from 1 to buffer size. The pacer is intended to be used in small-buffer networks of 20-30 packets, which is around 10-15 Kb on average. For the sake of consistency and ease of calculation, we store both packet size S_p and instantaneous queue size $q(t)$ in terms of 8-byte words, since this is the data width of the pipeline anyway. This results in a maximum packet size of 187 words for a 1500 byte packet. The delay ranges from a maximum of 20,000 cycles $max(d) = S_p/u_{min}$ for the MTU to a few hundred cycles for small size packets. Extrapolating these statistics brings us to a possible 200K to 300K entries with a memory size of 1 Mbits.

V. Evaluation

In this section, we evaluate the throughput and hardware overhead of implemented pacer. We compare our implemented pacing capable router against base reference router implementation that comes with NetFPGA. This gives us some insight on area overhead of employing a pacing inside a router.

A. Hardware Overhead

NetFPGA reference router's pipeline clock runs at 125 MHz. NetFPGA can be set to run at either 62.5 MHz or 125 MHz. In our initial straightforward implementation, the maximum achievable clock frequency was 59 MHz. After modifying the delay equation algebraically as explained in the previous section, the pacer could be run at 125 MHz.

Table I shows the resource usage in the reference router design that is part of the NetFPGA base package. In addition to that, it shows the overhead that the FPI implementation adds to the reference design, which is around 10%. LTI, on the other hand, uses even lesser resources due to precision being sacrificed for throughput and speed. Please note that the number of slices is not a meaningful metric in terms of area since it shows how optimally logic gates have been packed together.

Table II gives some insight about maximum achievable throughput and minimum possible latency. Since our floating divider takes 4 cycles to complete, and is not pipelined, maximum achievable throughput with FPI is 2 Gbps, where as with LTI that is 8 Gbps. We see an extra cycle of latency with LTI and 4 cycles of latency with FPI implementation.

Figure 7 demonstrates the pacing. It shows the dequeuing of the packets in reference router (non-paced) and in our pacing-capable router implementation (paced). Each point in this figure corresponds to the departure time of a packet. There are about 30 packets leaving the queue. It takes about 2 ns without pacer for all 30 packets leave the queue, while it takes about 4.5 ns for all 30 packets leave the queue when a pacer is installed. Without a pacer incoming packets leave the queue more bursty as it can be seen in the figure. However, when

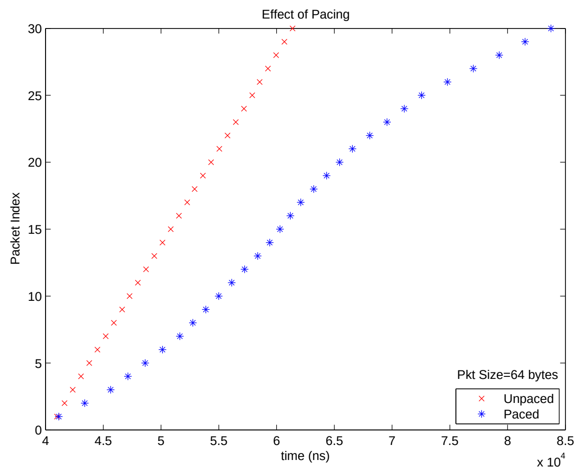

Fig. 7. Operation of Prototype Pacer.

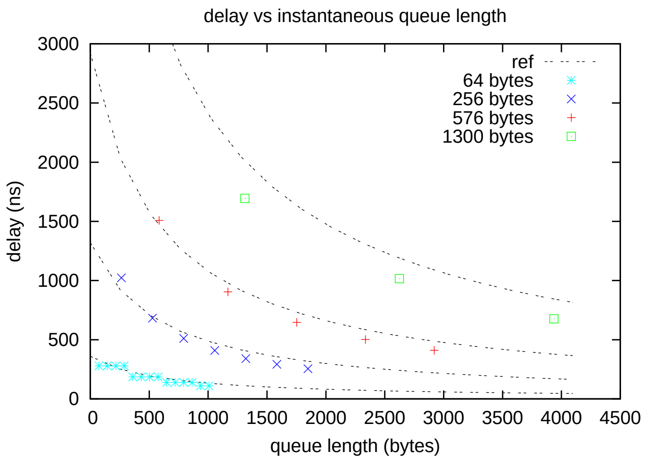

Fig. 8. Measured Delay and Reference Delay vs. Queue Length on Prototype Pacer.

the pacer is used the pacer delays the departure of the packet except the first one, and in turn smoothes the traffic.

Figure 8 shows the implemented LTI based pacer delay characteristics for different size packets and different queue lengths. Reference plot lines (shown as ‘ref’ in the figure) show the exact values and dots represent the delay values that are calculated by our LTI implementation. Since our LTI implementation uses interpolation to calculate delays of intermediate values, it sacrifices a little accuracy with respect to FPI implementation. This figure shows that if average size of the incoming packets is small then the error will be small regardless of the queue length.

VI. Summary and Conclusions

High-performance networks with small packet buffers represent a possible core technology for the future Internet. An important consideration is to ensure that traffic characteristics are suitable for these networks. The burstiness of traffic generated by the widely used transmission control protocol is particularly problematic for networks with small buffers. In prior work, we have shown that traffic pacing on nodes

TABLE I
RESOURCE UTILIZATION OF PROTOTYPE PACER ON NETFPGA PLATFORM.

	Reference Router		FPI			LTI		
Resource Type	Count	Utilization	Count	Utilization	Overhead	Count	Utilization	Overhead
Flip Flops	16,433	34%	17,180	36%	4.5%	16,552	35%	0.7%
4 input LUTs	22,954	48%	25,423	53%	10.8%	23,792	50%	3.7%
Block RAMs	106	45%	111	47%	4.7%	132	56%	24.5%
MULT18X18s	0	0%	6	2%	+6	0	0	0%
Slices	15,796	66%	17,530	74%	11.0%	15,514	65%	-1.8%

TABLE II
PERFORMANCE OF PROTOTYPE PACER.

	Reference Router	FPI	LTI
Maximum Block Latency (cycles)	1	4	1
Clock Frequency (MHz)	125	125	125
Theoretical Upper Limit (Gb/s)	8	2	8
Extra Latency (clock cycles)	0	4	1

inside the network can alleviate these problems and lead to higher network throughput. In this paper, we present the design and prototype implementation of a high-performance pacing system that can be implemented with low overhead on the output port of routers. We show that the design performs pacing as intended and that it does not degrade the performance of the router. We believe that this work presents an important step toward the efficient operation and wide-scale deployment of small-buffer networks, including optical packet-switched networks.

REFERENCES

[1] D. K. Hunter and I. Andonovic, "Approaches to optical Internet packet switching," *IEEE Communications Magazine*, vol. 38, no. 9, pp. 116–122, Sep. 2000.

[2] C. Qiao and M. Yoo, "Optical burst switching (OBS) – a new paradigm for an optical Internet," *Journal of High Speed Networks*, vol. 8, no. 1, pp. 69–84, Mar. 1999.

[3] S. Verma, H. Chaskar, and R. Ravikanth, "Optical burst switching: A viable solution for terabit IP backbone," *IEEE Network*, vol. 14, no. 6, pp. 48–53, Nov. 2000.

[4] R. Langenhorst, M. Eiselt, W. Pieper, G. Grosskopf, R. Ludwig, L. Kuller, E. Dietrich, and H. G. Weber, "Fiber loop optical buffer," *Journal of Lightwave Technology*, vol. 14, no. 3, pp. 324–335, Mar. 1996.

[5] J. Postel, "Transmission Control Protocol," Information Sciences Institute, RFC 793, Sep. 1981.

[6] V. Jacobson, "Congestion avoidance and control," in *Proc. of ACM SIGCOMM 88*, Stanford, CA, Aug. 1988, pp. 314–329.

[7] A. Vishwanath, V. Sivaraman, and M. Thottan, "Perspectives on router buffer sizing: recent results and open problems," *SIGCOMM Computer Communication Review*, vol. 39, pp. 34–39, Apr. 2009.

[8] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," *SIGCOMM Computer Communication Review*, vol. 34, no. 4, Oct. 2004.

[9] A. Dhamdhere and C. Dovrolis, "Open issues in router buffer sizing," *SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 87–92, Jan. 2006.

[10] T. Wolf, W. Gong, and Y. Cai, "Burstiness as traffic metric in next-generation optical core networks," in *Proc. of IEEE Photonics Society Summer Topicals*, Newport Beach, CA, Jul. 2009, pp. 129–130.

[11] Y. Cai, Y. Liu, W. Gong, and T. Wolf, "Impact of arrival burstiness to queue length: An infinitesimal perturbation analysis," in *Proc. of 48th IEEE Conferences on Decision and Control (CDC)*, Shanghai, China, Dec. 2009.

[12] Y. Cai, B. Jiang, T. Wolf, and W. Gong, "A practical on-line pacing scheme at edges of small buffer networks," in *Proc. of the 29th IEEE Conference on Computer Communications (INFOCOM)*, San Diego, CA, Mar. 2010.

[13] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "NetFPGA–an open platform for gigabit-rate network switching and routing," in *MSE '07: Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education*, San Diego, CA, Jun. 2007, pp. 160–161.

[14] Y. Cai, Y. S. Hanay, and T. Wolf, "Practical packet pacing in small-buffer networks," in *Proc. of IEEE International Conference on Communications (ICC)*, Dresden, Germany, Jun. 2009.

[15] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden, "Part III: Routers with very small buffers," *SIGCOMM Computer Communication Review*, vol. 35, no. 3, pp. 83–90, Jul. 2005.

[16] G. Hasegawa, T. Tomioka, K. Tada, and M. Murata, "Simulation studies on router buffer sizing for short-lived and pacing TCP flows," *Computer Communications*, vol. 31, no. 16, pp. 3789–3798, 2008.

[17] V. Sivaraman, H. Elgindy, D. Moreland, and D. Ostry, "Packet pacing in small buffer optical packet switched networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 4, pp. 1066–1079, Aug. 2009.

[18] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the performance of TCP pacing," in *Proc. of IEEE INFOCOM 2000*, Tel Aviv, Israel, Mar. 2000, pp. 1157–1165.

[19] O. Alparslan, S. Arakawa, and M. Murata, "Node pacing for optical packet switching," in *Proc. of the International Conference on Photonics in Switching*, Aug. 2008, pp. 1–2.

[20] "NetFPGA," http://www.netfpga.org/.