High-resolution Timer-based Packet Pacing Mechanism on the Linux Operating System

Ryousei Takano, Tomohiro Kudoh, Yuetsu Kodama, Fumihiro Okazaki

Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology (AIST)
{takano-ryousei, t.kudoh, y-kodama, f-okazaki}@aist.go.jp

***Abstract*— Packet pacing is a well-known technique for reducing the short-time-scale burstiness of traffic, and software-based packet pacing has been categorized into two approaches: a timer interrupt-based approach and a gap packet-based approach. The former was hard to implement for Gigabit class networks because it requires the operating system to maintain a microsecond resolution timer per stream, thus incurring a large overhead. On the other hand, a gap packet-based packet pacing mechanism achieves precise pacing without depending on the timer resolution. However, in order to guarantee the accuracy of rate control, the system had to have the capability to transmit packets at the wire rate. In this paper, we propose a high-resolution timer-based packet pacing mechanism that determines the transmission timing of packets by using a sub-microsecond resolution timer. With recent progress in hardware protocol offload technologies and multicore-aware network protocol stacks, we believe high-resolution timer-based packet pacing has become practical. Our experimental results show that the proposed mechanism can work on a wider range of systems without degradation of the accuracy of rate control. However, a higher CPU load is observed when the number of traffic classes increases, compared to a gap packet-based pacing mechanism.**

Keywords: packet pacing, high-resolution timer, TCP segmentation offload

I. Introduction

The transmission Control Protocol (TCP) is widely used on diversified computer networks, including the Internet, as well as inside data centers. Some researchers have reported TCP throughput collapse due to the TCP incast problem on data center networks [1][2] and traffic burstiness over long fat networks (LFNs) [3][4]. To address this issue, we need an appropriate and easy to use technique for network traffic control. Pacing is a well-known technique for reducing the short-time-scale burstiness of traffic. However, it is not often used. Some hardware based-pacing products have been deployed, such as the Chelsio network interface card (NIC), but such products are very rare on the market. On the other hand, software-based pacing was found difficult to implement for Gigabit class networks because it requires the operating system to maintain a microsecond resolution timer per TCP stream, thus incurring a large overhead.

To tackle this issue, we proposed PSPacer (Precise Software Pacer) [5] [6], which is a software module that achieves precise network bandwidth control and smoothing of bursty traffic without any special hardware. The key idea was to determine transmission timing of packets by the number of bytes transmitted. We call this a *byte clock*. If packets are transmitted back to back, the timing at which a packet is sent can be determined by the number of bytes sent before the packet. PSPacer fills the gaps between time aligned "real packets" (the packets which are sent by the user program) by "gap packets," as shown in Figure 1. The real packets and gap packets are sent back to back, and consequently, the timing of transmission of each real packet can be precisely controlled by adjusting the gap packet size. PSPacer can control the inter packet gap with 8 nanosecond resolution for Gigabit Ethernet, because 8 nanoseconds is taken to transmit 1 byte of data. The IEEE 802.3x PAUSE frames are used as gap packets. PAUSE frames are discarded at a switch input port, and only real packets go through the switch, keeping the original intervals.

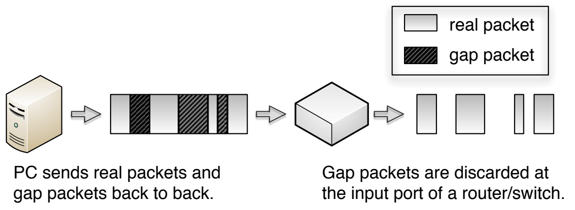

real packet
gap packet
PC sends real packets and gap packets back to back.
Gap packets are discarded at the input port of a router/switch.

Fig. 1. Gap packet.

However, PSPacer has the following limitations in general use. (1) PSPacer essentially requires the system to fully fill the network pipe with packets. Therefore, the system has to have the capability to transmit packets at the wire rate. For instance, if a Gigabit Ethernet NIC is connected through a 33MHz/32bit PCI bus, the bottleneck is thus PCI bus, and then the system can not transmit packets at 1 gigabit per second. As a result, the output traffic becomes imprecise. (2) PSPacer uses the IEEE 802.3x PAUSE frame as a gap packet. Therefore, PSPacer only works on Ethernet. And also, PAUSE frames can not be used for the original purpose. Therefore, the IEEE 802.3x flow control can not stop transmission from the switch and/or router to the PC. (3) PSPacer can not support pseudo network devices such as bonding and tap devices, because these devices can not properly handle gap packets.

In order to resolve the above limitations of the original PSPacer without degradation of the accuracy of rate control, we rethought the timer-based approach using a high-resolution timer, which employs nanosecond time stamps instead of using traditional millisecond resolution time ticks. A low-resolution timer event is scheduled to be activated at one of the periodic ticks, which involves several housekeeping tasks such as deferred interrupt processing. Therefore, the shorter the time periodic ticks the greater the CPU overhead. On the other hand, a high-resolution timer is a more light-weight mechanism, because it is dedicated to one purpose and it can be scheduled at arbitrary times independently of the periodic ticks. In addition, with recent progress in protocol offload technologies for the NIC, and multicore-aware network protocol stacks for the kernel, we believe the CPU overhead problem can be overcome. In this paper, we propose a high-resolution timer-based packet pacing mechanism called PSPacer/HT (High-resolution Timer), which controls transmission timing by using a high-resolution timer run according to the byte clock. This paper shows PSPacer/HT resolves the limitations of the original PSPacer, and addresses issues with the CPU load.

The rest of the paper is organized as follows. Section II shows the details of byte clock scheduling and our implementation using a high-resolution timer. In Section III, we describe an experiment methodology for evaluating the accuracy of packet pacing mechanisms. The experimental results are shown in Section IV. In Section V, we briefly mention related work on implementations and applications for packet pacing. Finally, Section VI summarizes the paper.

II. High-resolution Timer-based Packet Pacing Mechanism

A. *Software packet pacing and recent network interface cards*

Ethernet NICs have evolved in speed from 10 megabits per second to 10 gigabits per second. In addition, recent NICs provide several hardware offload mechanisms to reduce the CPU load by offloading protocol processing onto the NIC. We discuss below our concerns with how packet pacing is realized on these NICs.

The first concern is that a finer granularity is required as the link speed gets faster. For instance, to regulate the transmission rate to one half of the physical link speed, the inter packet gap must have the same size as the preceding packet after every packet. The upper half of Figure 2 shows a packet-level traffic diagram of identical pacing on a 10 gigabit link, where the target rate is 5 Gbps and the MTU size is 9 KB. The inter packet gap is precisely 7.2 microseconds. Likewise, when the MTU is 1.5 KB, the inter packet gap is 1.2 microseconds. Therefore, a sub-microsecond resolution timer is required for achieving packet pacing on 10 Gigabit links.

The second concern is TCP Segmentation Offload (TSO). TSO enables software (i.e., protocol stack, device drivers, etc) to send a pseudo large packet called *a TSO packet*, whose size is usually up to 64 KB and is independent of the MTU size. When the MTU size is set to 9 KB, the NIC divides the TSO packet into, at most, 7 packets. That means 7 packets are transmitted in burst when TSO is enabled, as shown in the

TABLE I
CPU utilization and goodput with configuration combinations of TCP segmentation offload and generic segmentation offload.

TSO	GSO	CPU util.	Goodput
enabled	enabled	8.56 %	9631.86 Mbps
disabled	enabled	13.21 %	9650.54 Mbps

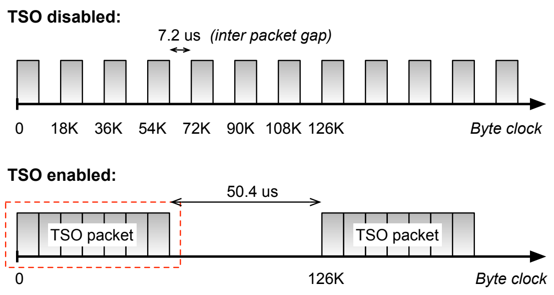


Fig. 2. Accuracy of pacing influenced by TCP segmentation offload.

bottom half of Figure 2. To realize the precise pacing shown in the upper half of Figure 2, TSO has to be disabled. In addition, the recent Linux kernel provides software-only segmentation offloading, called Generic Segmentation Offload (GSO), in which the segmentation processing is postponed from the protocol stack to the device driver.

Table I shows CPU utilization and goodput (an actual application level throughput) with configuration combinations of TSO and GSO via 10 Gigabit Ethernet on Linux.[1] TSO provides a significant reduction in CPU utilization. If TSO or GSO is enabled, transmission processing is distributed across multiple CPUs. In these cases, 4 CPU cores are utilized. On the other hand, if both TSO and GSO are disabled, goodput degrades to only 8 Gbps. This is because only 1 CPU core is utilized for transmission processing.

On the basis of the above discussion, we consider how much scheduling granularity is required for practical use. TSO introduces a trade-off between the accuracy of rate control and the CPU load. If a switch has enough buffer capacity per port to store a TSO packet (i.e., 64 KB), performance degradation due to buffer overflow does not occur. We think this size buffer is a reasonable assumption. Therefore, TSO may not introduce the performance issue. We need to develop a packet pacing mechanism coexistant with TSO.

B. *Byte clock scheduling*

We have been developing PSPacer to achieve precise packet pacing without any special hardware. The key to PSPacer is determining the transmission timing of packets by the number of bytes transmitted. If packets are transferred back to back,

[1]The experimental setting is that of PC A shown in Table 3. Note that CPU utilization of 100% means 8 processor cores fully utilized.

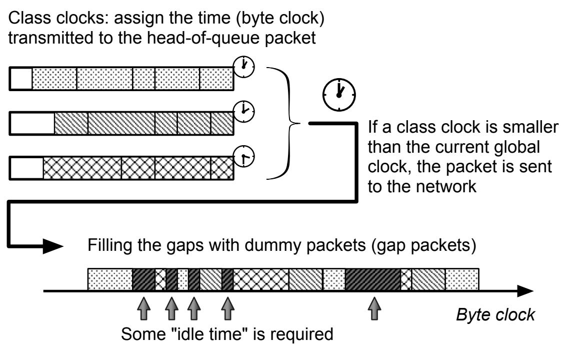

Class clocks: assign the time (byte clock) transmitted to the head-of-queue packet
If a class clock is smaller than the current global clock, the packet is sent to the network
Filling the gaps with dummy packets (gap packets)
Byte clock
Some "idle time" is required

Fig. 3. Byte clock scheduling.

the transmission timing of a packet is determined by the number of bytes sent before the packet.

Figure 3 shows an overview of byte clock scheduling. There are two type of byte clocks: a global clock and a class clock. A global clock keeps the number of bytes transmitted from a network interface. Outgoing packets are classified into classes according to filtering rules (e.g., source or destination IP address, port number, etc). Each class has a class clock which assigns the time trasmitted to the head-of-queue packet.

The packet scheduler selects a class in ascending order of the class clock. If the class clock is smaller than the global clock, it is time to transmit the head-of-queue packet of the class. Otherwise, no packet is transmitted.

When a packet is transmitted, the global clock increases by $t(s)$, where $t(s)$ is the time taken to transmit a packet whose size is s. At the same time, the class clock increases by (s, r), where r is the target transmission rate, and (s, r) is the delta time between the current clock and the next packet transmission on the class queue.

In byte clock scheduling, $t(s)$ is equal to the packet size s because the byte clock is the number of bytes transmitted. (s, r) is calculated as follows:

$$(s, r) = \frac{r_{max}}{r} \times s \qquad (1)$$

where r_{max} is the maximum transmission rate (i.e., the physical bandwidth of the network interface). Equation 1 means the bandwidth is controlled according to the ratio of the target bandwidth to the maximum transmission bandwidth of the system. Note that if the packet is a TSO packet, s is added to the size of IP and TCP headers generated by the NIC.

C. Timer subsystem on the Linux kernel

Before showing the implementation of PSPacer, we will briefly introduce the timer subsystem of the Linux kernel. The timer subsystem provides two types of timer mechanisms: a low-resolution timer and a high-resolution timer. In particular, we feel the latter type, which is supported by recent operating systems like Linux, is suitable for packet scheduling.

Figure 4 shows the comparison between low- and high-resolution timer events. The timer tick is a periodic timer interrupt that is usually generated HZ times per second, with the value of HZ varying between 100 to 1000, which depends on the kernel configuration. The timer ticks are also called *jiffies* in the Linux kernel. A low-resolution timer event is scheduled to be activated at one of the jiffies. In contrast, a high-resolution timer is set to be activated at the next required event time. A high-resolution timer employs nanosecond time stamps instead of using millisecond-resolution jiffies. The high-resolution timer event is activated when an interrupt occurs on the clock event device such as the CPU-local Advanced Programmable Interrupt Controller (APIC) timer. On the Intel IA32 architecture, the default clock source and the clock event device are the Time Stamp Counter (TSC) and the local-APIC timer, respectively[2]. In that case, the high-resolution timer mechanism uses the local APIC timer with the one-shot mode.

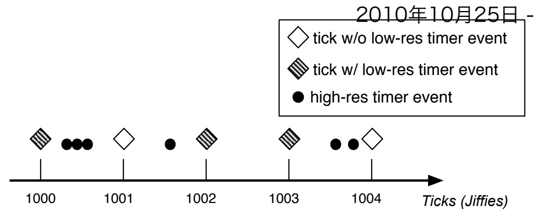


Fig. 4. Low- and high-resolution timer events.

In Linux kernel 2.6.21 and earlier, the packet scheduling subsystem used the low-resolution timer with a jiffies resolution. In Linux kernel 2.6.22 and later, it was changed to use the high-resolution timer mechanism called *hrtimer*. After kernel 2.6.31, the timer resolution improved from 1 microsecond to 1/16 microsecond.

D. PSPacer and PSPacer/HT

The original PSPacer [5] [6] uses gap packets to control the transmission interval between packets. PSPacer fills gaps between real packets (i.e., “idle time”) with gap packets. The real packets and gap packets are sent back to back. Gap packets are discarded at a switch input port, and only real packets go through the switch, keeping the original intervals. As the gap packets on Ethernet, the IEEE 802.3x PAUSE frames [7] with the pause time of zero are used. However, PSPacer has some limitations in general use. The system (computer, network interface, operating system, buffer settings, etc.) requires the capability to transmit packets at the maximum transmission rate (i.e., 10 Gbps for 10 Gigabit Ethernet, 1 Gbps for Gigabit Ethernet). PSPacer can not support pseudo network devices such as bonding and tap devices.

We implemented a high-resolution timer-based PSPacer called PSPacer/HT. PSPacer/HT does not have the above limitation of the original PSPacer. PSPacer/HT controls inter packet gaps by using a high-resolution timer instead of inserting gap packets. PSPacer/HT determines the transmission

[2]Several clock sources, including Jiffies, Programable Interrupt Timer (PIT), High Precision Event Timer (HPET), and TSC, are available, depending on the platform. If a low-resolution clock source such as Jiffies is chosen, the behavior of a high-resolution timer becomes similar to that of a low-resolution timer.

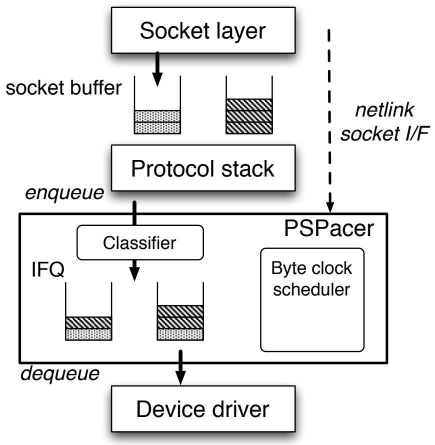


Fig. 5. Implementation of PSPacer.

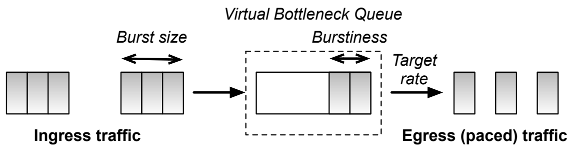


Fig. 6. Burstiness Model.

TABLE II

THEORETICAL BURST SIZE AND BURSTINESS (MAXIMUM BANDWIDTH 10 GBPS, TARGET BANDWIDTH 5 GBPS, AND MTU 9 KB).

	burst size	inter-burst time	maximum burstiness
high-res timer without TSO	1	7.2 us	1
high-res timer with TSO	7	50.4 us	4
low-res timer (1 ms)	70	504 us	35

timing of a packet and then registers the timing as a high-resolution timer event. When the event is activated, the packet is transmitted. PSPacer/HT can work with pseudo network devices such as bonding and tap devices. This is because PSPacer/HT does not need to handle gap packets.

Figure 5 shows an overview of the implementation of PSPacer. Basically, both PSPacer and PSPacer/HT are implemented as Queueing Discipline (QDisc) modules. QDisc provides an algorithm that manages interface queues (IFQs) of a corresponding NIC device. Currently, PSPacer is a classfull queue which contains a multiple classes; PSPacer/HT is a classless queue which contains single IFQ and no classifier. We designed PSPacer/HT to be used as a leaf QDisc of a classful QDisc. Of cource, PSPacer/HT can be implemented as a classful QDisc. Users can configure parameters such as the target rate and the filter rules of classification by means of iproute2 `tc` command, which communicates with PSPacer via a netlink socket interface.

III. METHODOLOGY

A. Definition of burstiness

A quantitative criterion is required for evaluating the accuracy of packet pacing. The burstiness of traffic is suitable for this purpose, because paced traffic minimizes the burstiness. In both cases mentioned in Figure 2, for instance, the average bandwidths are the same, but TSO increases short-time-scale burstiness. In this paper, we define burstiness as the queue size of a bottleneck queue, as shown in Figure 6. This definition is based on the fact that packets are queued, and then the difference between ingress and egress rates is directly reflected in the queue length, i.e., the amount of data stored at the buffer before the bottleneck.

Our burstiness model assumes that the observed traffic goes through via a virtual bottleneck (VB) with a given bandwidth (*a target rate*) and an infinite queue (VBQ). The ingress traffic may consist of chunks of packets. We call the number of packets of a chunk its *a burst size*. If the ingress traffic exceeds the target rate, the excess traffic would be stored in a VBQ and then the length of the VBQ increases, while if it is below the target rate, the length decreases.

Assuming only one chunk of packets goes thought the virtual bottleneck queue, the instant burstiness is estimated as follows:

$$burstiness_i = \left\lceil burst\ size \times \frac{(max\ rate - target\ rate)}{max\ rate} \right\rceil \tag{2}$$

The unit of burstiness is the number of packets.

However, we need to consider a series of chunks, and both the interval and the burst size varies during transmission. The maximum value of $burstiness_i$ over a period of measurement time is called maximum burstiness.

Table II summarizes the relationship among burst size, inter-burst time, and maximum burstiness, where intervals between chunks are uniform. The maximum bandwidth is 10 Gbps, the target rate is 5 Gbps, and the MTU is 9 KB. The maximum burstiness of high-resolution timer-based pacing with TSO is about 10 times smaller than that of low-resolution timer-based pacing with 1 millisecond jiffies.

B. Experimental method

In this paper, we developed a methodology for off-line measurement of burstiness. This work is based on a real-time burstiness measurement method proposed in another paper [8]. Each experiment in the off-line method follows the steps below:

1) Setting the target rate and choosing pacing mechanisms: PSPacer, PSPacer/HT, HTB, and so on.
2) Capturing packets generated by a bulk transfer program like the Iperf benchmark.
3) Analysing burstiness using a network simulation with captured packets.

IV. EXPERIMENT

A. Experimental setting

We used two PC platforms. Each PC platform consisted of a pair of the same specification PCs: PC A or PC B, as

TABLE III
PC CLUSTER SPECIFICATIONS.

PC A	
CPU	Intel Quad-core Xeon E5430 2.66 GHz
Memory	8 GB DDR2-667
Ethernet(1)	Myricom Myri-10GE
Ethernet(2)	Intel Gigabit ET dual port (82576)
I/O Bus	PCI-Express x8
OS	ubuntu 9.10 Linux kernel 2.6.31-10
PC B	
CPU	Intel Xeon 3.0 GHz dual
Memory	1 GB DDR2-400
Ethernet	Intel PRO/1000 (82541GI)
I/O Bus	PCI 33MHz/32bit
OS	CentOS 5.3 Linux kernel 2.6.31.13

TABLE IV
SYSCTL PARAMETERS.

net.core.netdev_max_backlog	250000
net.core.wmem_max	16777216
net.core.rmem_max	16777216
net.ipv4.tcp_rmem	4096 87380 16777216
net.ipv4.tcp_wmem	4096 65536 16777216
net.ipv4.tcp_no_metrics_save	1

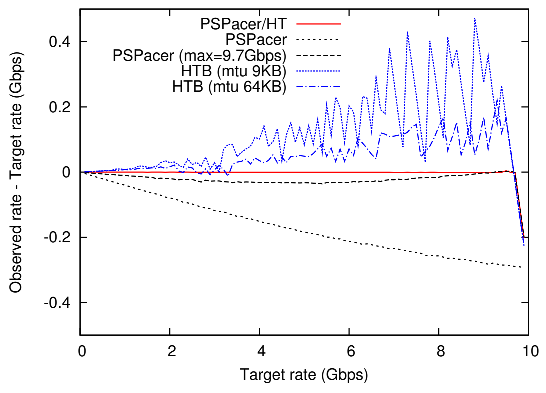


Fig. 7. Delta between the observed rate and the target rate.

TABLE V
MAXIMUM DELTA IN THE TRANSMISSION RATE AND THE ERROR RATE BETWEEN THE OBSERVED RATE AND THE TARGET RATE.

PSPacer/HT	+36 Kbps (0.0%)
PSPacer	-287 Mbps (-5.7%)
PSPacer (max=9.7Gbps)	-36 Mbps (-0.7%)
HTB (mtu=9KB)	+473 Mbps (+9.5%)
HTB (mtu=64KB)	+219 Mbps (+4.4%)

shown in Table III. PC A had two network interfaces: a 10 Gigabit Ethernet (Myricom Myri-10G) and a dual-port Gigabit Ethernet (Intel Gigabit ET dual port). Each Myri-10G port was connected through a 10 Gbps hardware network testbed called GtrcNET-10[9]. PC B had a Gigabit network interface, connected to a 33MHz/32bit PCI bus. This means that PC B could not transmit packets at the rate of gigabits per seccond due to the PCI bus bottleneck. PC A was used in all the following experiments; PC B was only used in the experiment in Section IV-E.

GtrcNET-10 consists of a large-scale Field Programmable Gate Array (FPGA), three 10 Gbps Ethernet XENPAK ports, and three blocks of 1 GB DDR-SDRAM. The FPGA is a Xilinx XC2VP100, which includes three 10 Gbps Ethernet MAC and XAUI interfaces. GtrcNET-10 provides many functions, such as traffic monitoring in microsecond resolution, traffic shaping, and WAN emulation at 10 Gbps wire speed. In the experiment, GtrcNET-10 was used to observe traffic monitoring and capture packets with a nanosecond-resolution time stamp complying with the RFC 1305 Network Time Protocol (NTP) format.

Each PC A was running the Ubuntu 9.10 server edition and Linux kernel 2.6.31-10. The device driver of Myri-10G was updated to version 1.5.1. Each PC B was running CentOS 5.3 and Linux kernel 2.6.31.13. Here we refer to two timer related kernel configurations. The kernel timer interval was set to 1 millisecond (i.e., CONFIG_HZ=1000), because of the accuracy of timer event handling. A more detailed explanation is described in Section IV-F. On that note, the dynamic ticks feature (i.e., CONFIG_NO_HZ), which allows for stopping the periodic tick when the kernel is idle or does almost nothing, was enabled. It caused no significant effect in this experiment. Some networking sysctl parameters were changed from the default value, as shown in Table IV, because default parameters, such as socket buffer sizes, are not adequate for a 10 Gigabit experiment. In addition, tcp_no_metrics_save disables reuse of the parameters of the previous connection.

Hierarchical Token Bucket (HTB), which is a variant of class-based queueing implementations [10], was used as a target for comparison. PSPacer/HT and HTB are based on the same high-resolution timer mechanism.

B. Average bandwidth

We confirmed that PSPacer/HT works properly over 10 Gigabit Ethernet. We measured the average bandwidth over a period of 5 seconds by using the Iperf benchmark. The target rate was set from 100 Mbps to 10 Gbps every 100 Mbps. Figure 7 shows the difference between the target rate and the average transmission rate observed by GtrcNET-10. The maximum differences in the transmission rate and the error rate is summarized in Table V.

With PSPacer, the difference becomes larger linearly as the target rate increases. The observed bandwidth is up to about 0.3 Gbps less than the target rate. This is because Myri-10G cannot achieve the wire rate speed in our experimental setting. The label 'PSPacer (max=9.7 Gbps)' indicates that the maximum transmission rate (r_{max} in Equation 1) is set not to 10 Gbps but 9.7 Gbps to calibrate the gap size calculation. In this case, the difference is almost negligible. In contrast, PSPacer/HT achieves an accurate bandwidth control on 10

Gigabit Ethernet, even if the system does not have enough capability to transmit packets at wire rate speed. On the other hand, HTB shows larger differences and wider variance than PSPacer/HT. The observed bandwidth is up to about 0.5 Gbps more than the target rate.

HTB controls the bandwidth in a stepwise manner, and so the delta shows a saw-tooth shape. The estimation error in the length to time lookup (`l2t`) table can be considered as the reason for this. The `l2t` table is used to determine how long it will take to transmit a packet given the size. QDisc modules, other than PSPacer and PSPacer/HT, pass the `l2t` table instead of the target rate from the `tc` command. The mtu parameter, which is independent of the physical MTU size, is used for calculating the `l2t` table. Setting a large mtu parameter is effective as shown in the label 'HTB (mtu 64KB)' case, but as a side-effect, some precision is lost for short packets because of the estimation error in `l2t` lookup. The `l2t` lookup table has only 256 slots. This paper does not discuss this problem in more detail.

C. Burstiness

We confirmed the burstiness of PSPacer, PSPacer/HT, PSPacer/HT with a low-resolution timer, and HTB, according to the methodology described in Section III. PSPacer/HT with a low-resolution timer emulates the behavior of the low-resolution timer by setting the clock source to Jiffies. The target rate is regulated at 5 Gbps. Using GtrcNET-10, we captured 700 thousand packets with a nanosecond resolution time stamp.

Table VI shows the results. The average burstiness of each simulation step is also shown just for comparison. The maximum burstiness is slightly larger than the theoretical values described in Table II. In the case where TSO is disabled, the maximum burstiness of PSPacer is 2, and this is the smallest value. In contrast, in the case where TSO is enabled, there is no significant difference among PSPacer, PSPacer/HT, and HTB. Switches and routers supporting 10 Gigabit Ethernet may be equipped with enough buffer capacity to have a tolerance for burstiness of less than 10 packets. Therefore, PSPacer/HT is able to avoid degradation of the accuracy of rate control at a sufficient level from a practical viewpoint.

Without TSO the size of all captured packets is the same as the MTU size. On the other hand, with TSO we observed no negligible short packets. The cause of this behavior is now under investigation. Anyway, with TSO, the buffer usage in a VBQ is smaller than burstiness × MTU, unlike without TSO.

To investigate the difference of burstiness in more detail, the transmission interval between 2 successive captured packets is plotted in Figure 8. Note that the Y-axis is logarithmic scale. When packets are transmitted in burst, the interval is 7.2 microseconds. In this case (the target rate is 5 Gbps), the theoretical interval is 14.4 microseconds. In the case of PSPacer without TSO (Figure 8 (1)), there is a peak at 14.4 microseconds, and no burst transmission. Some intervals spread up to 30 microseconds because the system does not have enough capability to transmit packets at wire rate speed,

TABLE VI
BURSTINESS FOR 4 IMPLEMENTATIONS: PSPACER (GAP PACKET BASED), PSPACER/HT (HIGH-RESOLUTION TIMER BASED), PSPACER/HT (LOW-RESOLUTION TIMER BASED), AND HTB. THE TARGET RATE IS 5 GBPS, AND MTU IS 9 KB.

	TSO	average burstiness	maximum burstiness
PSPacer	disabled	1.73	2
	enabled	3.08	7
PSPacer/HT	disabled	2.06	7
	enabled	3.27	9
HTB	disabled	2.00	7
	enabled	3.12	8
Low-res timer (1ms)	enabled	17.5	39
Low-res timer (10ms)	enabled	98.3	534

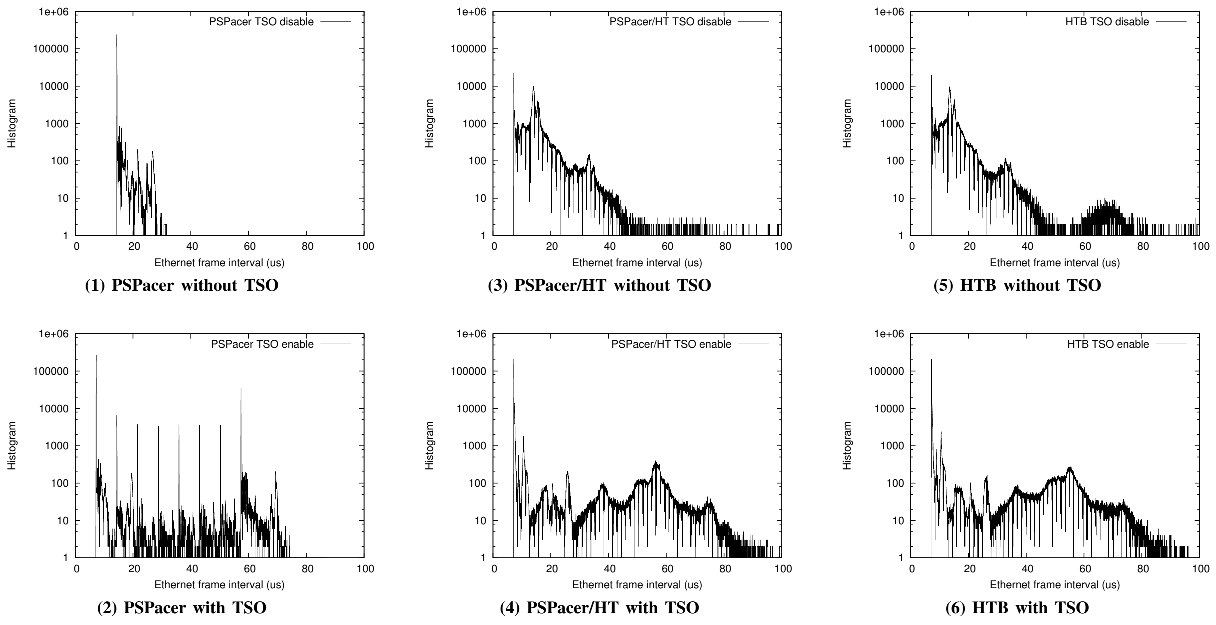
as described above. In the case of PSPacer with TSO (Figure 8 (2)), there are 8 spikes at a multiple of 7.2 microseconds, because the interval is proportional to the TSO packet size. In the cases of PSPacer/HT (Figure 8 (3), (4)) and HTB (Figure 8 (5), (6)), there are no significant differences. This is because both are implemented based on the same high-resolution timer mechanism. Here it should be noted that even if TSO is disabled these two mechanisms transmit packets in burst. This supports the increase in burstiness of both PSPacer/HT and HTB.

D. CPU load

The operating system maintains a high-resolution timer per TCP flow, which could incur lots of overhead. To evaluate the CPU overhead of high-resolution timer-based pacing as compared to gap packet-based pacing, we measured the CPU load while varying the number of traffic classes. Each traffic class is set to be a one-to-one mapping with a single flow at the same time. The CPU load is measured by the `iostat` command.

Table VII shows the CPU utilization. Note that "CPU utilization is 100%" means 8 processor cores fully utilized. Each column indicates the target rate per stream setting: the same as the total target rate, 100 Mbps, and 50 Mbps. In the case of the setting when the total target rate is 1 Gbps and the target rate per stream is 100 Mbps, the number of streams is 10. This means the traffic distributes 10 traffic classes whose target rate is 100 Mbps.

In both PSPacer/HT and HTB, the CPU load increases more rapidly as the number of traffic classes increases as compared with PSPacer. This is because of the overhead of timer handling with higher frequency. Comparing PSPacer/HT with HTB, the load is lower when the total target rate is 8 Gbps. However, the CPU load is higher when the total target rate is 4 Gbps and lower. The reason is under investigation. In order to reduce the CPU load, several timer events can be combined with a trade-off between the CPU load and the accuracy of packet scheduling in mind. It is an open issue.



(1) PSPacer without TSO **(3) PSPacer/HT without TSO** **(5) HTB without TSO**

(2) PSPacer with TSO **(4) PSPacer/HT with TSO** **(6) HTB with TSO**

Fig. 8. Histograms of packet transmission interval.

TABLE VII
CPU UTILIZATION AND THE NUMBER OF STREAMS.

Total target rate	PSPacer Target rate	PSPacer 100 Mbps	PSPacer 50 Mbps	PSPacer/HT Target rate	PSPacer/HT 100 Mbps	PSPacer/HT 50 Mbps	HTB Target rate	HTB 100 Mbps	HTB 50 Mbps
1 Gbps	0.66	0.94	1.04	0.71	0.81	0.91	0.84	0.76	0.82
2 Gbps	1.80	2.12	2.16	1.60	1.93	2.44	1.83	1.72	1.88
4 Gbps	3.74	4.29	4.78	3.66	5.81	8.19	3.92	4.15	4.49
8 Gbps	7.67	10.11	11.19	8.35	11.09	17.04	8.88	12.32	25.55

Each column indicates the target rate per stream: target rate, 100 Mbps, and 50 Mbps.
The label 'target rate' means the number of streams is 1.

E. Operation check on a wider rage of systems

We confirmed that PSPacer/HT works properly with Gigabit Ethernet connected thought a 32MHz/32bit PCI bus (PC B in Table III) and dual port Gigabit Ethernet bonding link (PC A Ethernet (2) in Table III).

Table VIII shows the goodput of the Iperf benchmark on PC B. In this setting, the maximum goodput is around 810 Mbps because of the bottleneck of the PCI bus. The theoretical value is calculated by $((MTU - sizeof(IP\ hdr))/(MTU + sizeof(Ethernet\ hdr))) \times target\ rate$, where MTU is 1500, and both $sizeof(IP\ hdr)$ and $sizeof(Ethernet\ hdr)$ are 20. PSPacer underestimates the target rate by from 8 % to 10 %. In contrast, the result of PSPacer/HT is identical with the theoretical value.

F. Timer event accuracy of the packet scheduler

PSPacer/HT determines the transmission timing of a packet and registers its timer event that is going to activate at a required point of time. When the high-resolution timer is fired, the packet is transmitted. The accuracy of generating timer events is important to achieve precise packet scheduling.

TABLE VIII
COMPARISON OF GOODPUT ON PC B (GIGABIT ETHERNET CONNECTED THROUGH A 32MHZ/32BIT PCI BUS).

Target rate	Theoretical	PSPacer	PSPacer/HT
500 Mbps	487 Mbps	448 Mbps	487 Mbps
600 Mbps	584 Mbps	535 Mbps	584 Mbps
700 Mbps	682 Mbps	620 Mbps	681 Mbps
800 Mbps	779 Mbps	707 Mbps	780 Mbps

We observed the difference in the time between setting of a timer event and activating a timer event handler. Here we call the difference *timer event latency*. Figure 9 shows the results with a kernel timer interval of 1 or 10 milliseconds. In both cases, the timer events are delayed up to the kernel timer interval. The larger the timer event delay, the larger its burstiness. With a 1 millisecond interval, the burstiness is 9 as shown in Table VI. On the other hand, with a 10 millisecond interval, the burstiness is increased to 53. These results show that the kernel timer interval is an important parameter, even if a high-resolution timer is employed. Therefore, we set the

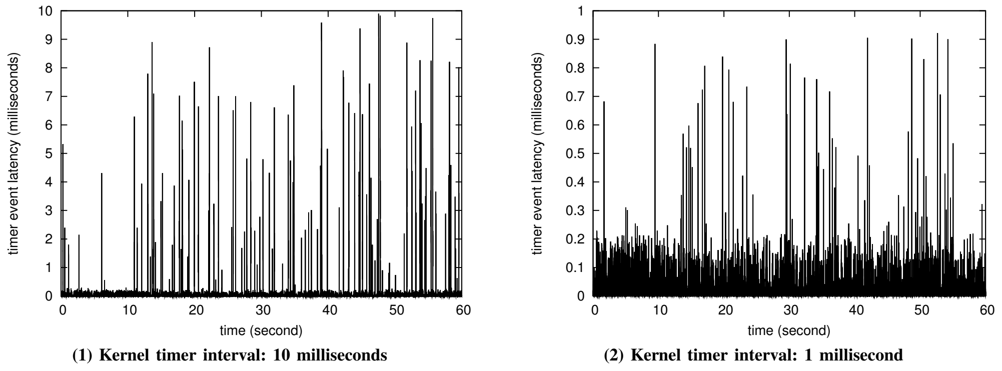


Fig. 9. Timer event latency of PSPacer/HT.

kernel timer interval to 1 millisecond.

V. Related Work

Many burstiness mitigation schemes have been proposed. Blanton, et al., [11] showed that packet losses caused by micro-bursts are not frequent on the Internet since the size of bursts is modest, while larger bursts increase the probability of packet losses. Jiang, et al., [12] investigated the fact that burstiness in sub-RTT scales can be significantly reduced by TCP pacing [13] if a kernel timer has sufficiently fine resolution. Recently some researchers have reported a catastrophic TCP throughput collapse, known as the TCP incast problem [1][2], that occurs as many nodes send to a single node. This phenomenon has been observed in distributed storage, HPC parallel application, and MapReduce workloads. Packet pacing may help the performance improvement for this kind of problem.

Hardware offloading mechanisms are required for achieving higher performance as the network speed becomes faster. TCP offloading hardware assists are classified in terms of the following two factors: stateful or stateless, and sender side or receiver side.

TCP Segmentation Offload (TSO) and checksum offload are stateless TCP offloading hardware assists on the sender side. On the other hand, a TCP Offload Engine (TOE) (e.g., Chelsio T210) is a stateful technology designed to offload all TCP/IP protocol processing to the NIC. These hardware assists provide a significant reduction in CPU utilization. Linux kernel developers are opposed to supporting TOE because of reasons described in [14]. For instance, TOE does not support many useful Linux networking features, including netfilter and iproute2.

On the receiver side, some Linux NIC drivers support Large Receive Offload (LRO) [15]. LRO aggregates incoming packets into fewer but larger packets. Reduction of the number of packets provides similar effects to those of TSO. A. Menon, et al., reported that as ‘per-byte’ operations, including data copying and checksumming, become cheaper due to hardware prefetching, ‘per-packet’ operations, including header processing and buffer management, become the dominant performance factor in TCP receive side processing [16].

We require a pacing-aware or -friendly hardware offloading mechanism. Chelsio TOE NICs provide micro-second granularity TCP pacing at a 10 Gigabit rate. Some NICs have a function to set the inter frame gap by writing a requested value into a certain register. T. Yoshino, et al., reported that precise pacing using this type of functionality is effective in optimizing performance of TCP/IP over 10 Gigabit Ethernet [4]. However, the method cannot control the transmission rate per stream. The controllable range of the inter frame gap is limited, depending on the NIC, and it usually is not sufficient. K. Kobayashi proposed a transmission timer approach for rate-based pacing with a little hardware support [17]. In this approach, the host software specifies the time that each packet should be transferred, and the NIC inserts a precise inter frame gap for the data stream. A combination of PSPacer and this type of NIC, which controls an inter frame gap without transmitting a gap packet, can be considered as effective.

VI. Conclusion and Future Work

This paper has proposed a high-resolution timer-based packet pacing mechanism, and shown it can work on a wide range of systems without degradation of the accuracy of rate control. Unlike a gap-packet based mechanism, the proposed mechanism does not require the capability to transmit packets at the physical link speed of the system, and supports pseudo devices such as bonding and tap devices. The experimental results show the proposed mechanism precisely controls the transmission rate in several network settings: 10 Gigabit Ethernet, Gigabit Ethernet connected thought a 32MHz/32bit PCI bus, and a dual port Gigabit Ethernet bonding link. However, higher CPU load is observed when the number of traffic classes increases, as compared with a gap packet-based packet pacing mechanism. As future work, we plan to develop a new scheduling technique to optimize the trade-off between the burstiness and the CPU load. In order to mitigate high-resolution timer interrupts, several timer events can be combined by considering the tolerance for burstiness on switches and routers.

This paper also has shown some problems in the HTB of the Linux kernel: lack of accurate rate control and the difficulty of parameter tuning. In order to improve the accuracy as well as keeping it easy to gain the requested transmission rate, reimplementing HTB based on the byte clock method is future work. It might be of great benefit to many Linux users.

ACKNOWLEDGEMENTS

This research was partially supported by both the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan, Grant-in-Aid for Young Scientists (20800083), and the New Energy and Industrial Technology Development Organization (NEDO) research project entitled "Research and Development Project for Green Network/System Technology (Green IT Project)"

REFERENCES

[1] A. Phanishayee, E. Krevat, V. Vasudevan, D.G. Andersen, G.R. Ganger, G.A. Gibson, and S. Seshan, "Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems," USENIX FAST 2008, February 2008.
[2] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D.G. Andersen, G.R. Ganger, G.A. Gibson, and B. Mueller, "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication," ACM SIGCOMM 2009, August 2009.
[3] O. Tatebe, H. Ogawa, Y. Kodama, T. Kudoh, S. Sekiguchi, S. Matsuoka, K. Aida, T. Boku, M. Sato, Y. Morita, Y. Kitatsuji, J. Williams, and J. Hicks, "The Second Trans-Pacific Grid Datafarm Testbed and Experiments for SC2003," IEEE/IPSJ SAINT 2004 Workshops, pp.26–30, January 2004.
[4] T. Yoshino, Y. Sugawara, K. Inagami, J. Tamatsukuri, M. Inaba, and K. Hiraki, "Performance Optimization of TCP/IP over 10 Gigabit Ethernet by Precise Instrumentation," SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, November 2008.
[5] R. Takano, T. Kudoh, Y. Kodama, M. Matsuda, H. Tezuka, and Y. Ishikawa, "Design and Evaluation of Precise Software Pacing Mechanisms for Fast Long-Distance Networks," PFLDNet 2005, February 2005.
[6] R. Takano, T. Kudoh, Y. Kodama, M. Matsuda, Y. Ishikawa, and F. Okazaki, "Precise software pacing method using gap packets," IPSJ Transactions on Advanced Computing Systems, vol.47, no.SIG 7 (ACS 14), pp.194–206, 2006.
[7] IEEE Std. 802.3x, "Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specification, Section 2 (Annex 31B)," 2005.
[8] R. Takano, Y. Kodama, T. Kudoh, M. Matsuda, and F. Okazaki, "Realtime Burstiness Measurement," PFLDNet 2006, February 2006.
[9] GtrcNET-10. `http://projects.itri.aist.go.jp/gnet/`.
[10] S. Floyd and V. Jacobson, "Link-sharing and Resource Management Models for Packet Networks," IEEE/ACM Transactions on Networking, vol.3, no.4, pp.365–386, 1995.
[11] E. Blanton and M. Allman, "On the Impact of Bursting on TCP Performance," Passive and Active Measurement Workshop 2005, March 2005.
[12] H. Jiang and C. Dovrolis, "Why is the Internet traffic bursty in short time scales?," ACM SIGMETRICS 2005, June 2005.
[13] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the performance of TCP pacing," IEEE INFOCOM 2000, pp.1157–1165, March 2000.
[14] The Linux Foundation, "TCP Offload Engine." `http://www.linuxfoundation.org/collaborate/workgroups/networking/toe`.
[15] L. Grossman, "Large Receive Offload implementation in Neterion 10GbE Ethernet driver," Linux Symposium 2005, July 2005.
[16] A. Menon and W. Zwaenepoel, "Optimizing TCP Receive Performance," USENIX 2008, pp.85–98, 2008.
[17] K. Kobayashi, "Transmission Timer Approach for Rate Based Pacing TCP with Hardware Support," PFLDNet 2006, February 2006.