End-node transmission rate control kind to intermediate routers - towards 10 Gbps era

Makoto Nakamura, Junsuke Sembon, Yutaka Sugawara,
Tsuyoshi Itoh, Mary Inaba and Kei Hiraki
University of Tokyo
{makoto, bonse, sugawra, tsuyoshi, mary, hiraki}@is.s.u-tokyo.ac.jp

February 15, 2004

Abstract

While transferring data using TCP/IP, even though total data transfer rate is same, microscopic behavior of data transmission is quite different by network interfaces of end nodes. Inappropriate data transmission rate causes needless packet losses which result in bad perforamance, and worse, it also causes needless loads for intermediate routers. This problem will become more serious as bandwidth becomes wider. This paper explains the problem and describes our approaches.

1 Introduction

It is well known that TCP/IP data transfer on LFN (Long Fat pipe Network) is difficult. The transfer rate is about $windowsize/RTT$, hence, LFN demands large window size. On the other hand, window size grows by ACK which returns after RTT interval, thus, growing speed of window size on LFN is slow, proportional to RTT. First window size control of current TCP is that window size is multiplicatively increased during slow start, and then "Additive Increase and Multiplicative Decrease (AIMD)" during congestion avoidance. For stationary state of single stream during congestion avoidance, a line graph of AIMD (throughput over time) forms saw-tooth shape, whose upper and lower bend-point is available bandwidth and $1/2$ of available bandwidth respectively. If AIMD works ideally in sufficiently long period, $3/4$ of available bandwidth should be utilized on the average. A lot of studies have been done for appropriate window size control, such as Fast TCP [15], Scalable TCP [14], and High-Speed TCP [13], and some theoretical studies have been done for analysis of AIMD [5, 6].

TxQ	100 packets			25000 packets		
	Min	Max	Middle	Min	Max	Middle
GbE	9.07	120.0	11.6	9.52	78.0	44.6
FE	25.6	25.7	25.6	88.6	88.7	88.7

Figure 1: GbE vs. FE on LFN, RTT 200 msec, bottleneck OC-12

But the low performance of LFN is not only induced by window size control. Long distance streams fail to grow their window size although network not seems to be congested and there exist sufficient time. Total utilization of network bandwidth is far from $3/4$. In addition, by changing network interface from Gigabit Ethernet (GbE) to Fast Ethernet (FE), performance can improve remarkably [20].

Table 1 shows the comparison of GbE and FE of data transfer between Maryland and Tokyo where bottleneck is OC-12 and RTT is 200 msec. We take data 11 times for GbE. The performance of GbE is very unstable. Sometimes, it attains over 100 Mbps, but, in many cases, FE shows better performance. This disperseness is critical because, in most cases, the slowest stream is the dominant factor of total performance.

Since the only difference is speed of *interface*, let us consider about speed more. We call $window\ size/RTT$ "flow-level rate" which is decided in transport layer. This flow-level rate is too macroscopic when we consider data transfer on LFN with high speed interface such as Gigabit Ethernet (GbE). 1500-bytes packets can be transmitted every 12.5 μs by GbE interface. We call this microscopic rate as "packet-level rate".

Let X denotes macroscopic flow-level rate and Y packet-

Figure 2: Transfer Rate is fixed $X = window\ size/RTT$, which Window Size Control Algorithm might expect

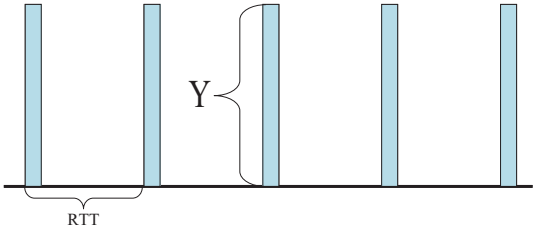

Figure 3: Real Transfer Rate change with microscopic view. Rate is fixed Y for $\Delta t = X \times RTT/Y$ of every RTT

level rate of the network interface. While transferring data, roughly speaking, the packet is sent via interface for only first $\Delta t = X \times RTT/Y$ of every RTT, which we call "busy Δt", then next $RTT - \Delta t$ time, that interface is idle. Hence, network needs buffer of size $Z = (Y - X) \times \Delta t$. This peaky behavior for busy Δt easily causes needless shortage of buffer memories of intermediate routers which results in needless packet losses. Note that busy Δt is long when RTT is long, and buffer size Z needs to be large when transmission rate Y is large and RTT is long. This problem becomes more serious when 10 Gbps interface is available for high-end PCs.

Fig.2 shows the ideal transfer rate which might be expected by window control algorithms. On the other hand, real transfer rate with microscopic view is like Fig.3. This pattern is formed mainly in Slow Start Phase, where 2 packets are sent just after one ACK is received repeatedly.

In this paper, our target is to approximate Fig.3 to Fig.2 with small overhead. This prevents the intermediate routers be overloaded, decreases packet drop and achieves high performance. For this purpose, we take 3 approaches.

IPG tuning Expanding busy Δt by enlarging space between Ethernet frames. Fig. 4 shows the expected IPG tuning result; lowering the packet-level rate. Bandwidth changes into $YF/(F + IPG)$ where F is the size of ethernet frame, (mostly, MTU) and IPG is the size of space between ethernet frame, and busy Δt becomes $\Delta t(F + IPG)/F$.

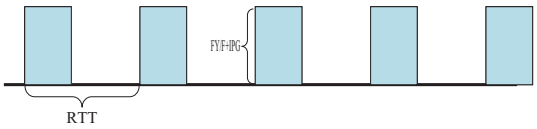

Figure 4: Packet Spacing by setting IPG

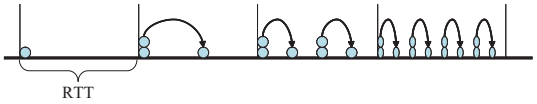

Figure 5: Clustered packet Spacing, First 3 RTTs of Slow Start Phase

Clustered Packet Spacing To split busy Δt into small pieces, putting an appropriate transmission interval between packets during slow start in TCP stack using kernel timer. When an ACK returns, sender sends one packet immediately, but wait sending another packet for half of expected interval of ACK arrivals until the precision of kernel timer permits. After that, normal TCP congestion control takes over. If we can control the transmission interval up to 1 ms, the size of splitted burst of data is limited to under 80 packets in case of GbE using 1,500-byte frames. Fig. 5 shows first 4 RTTs to reclamate the basin between busy Δt s.

TCP-aware NIC Directly realize Fig.2 using hardware support. NIC is informed both RTT and $window\ size$ and dynamically fix its Ethernet frame interaval.

IPG tuning and Clustered Packet Spacing is independent or rather supplements each other, since IPG tuning lowers the peak but busy Δt still remains until $(F + IPG)/F = Y/X$. On the other hand, Clustered Packet Spacing works as dividing "$RTT - \Delta t$" of idle time. Both IPG tuning and Clustered Packet Spacing is controlling idle and busy time of the interface statistically. These are software solution and experiments are proceeded between Tokyo, Japan and Maryland, U.S., 7200 miles distance with bottle neck OC12/POS. We use `iperf` command on DELL PowerEdge 1650 (Dual Pentium-III 1.4 GHz, 1 GB memory, on-board NIC with Intel 82554 chip) with Linux 2.4.22 and 2.6.0-test5 with NIC driver e1000 version 5.1.13, without TCP Segmentation Offload functionality. GbE and FE are configured by Extreme Summit5i switch.

TCP-aware NIC is, in some sense, our final goal, which

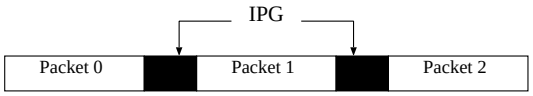

Figure 6: Inter Packet Gap

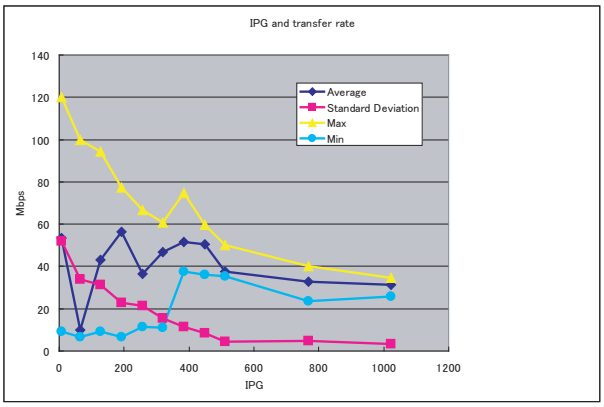

Figure 7: IPG (8-1023 bytes) graph

proceeds the tuning dynamically; i.e., it changes its packet intervals as window size and RTT changes. We show its basic design and simulation result.

2 Inter Packet Gap tuning

Inter Packet Gap (IPG) is a gap which is placed between Ethernet frames to make transmitter be idle (Fig. 6). Its value should be more than 12 bytes (by IEEE 802.3), but default IPG of ethernet driver e1000 is settled to 8 bytes. By modifying e1000 ethernet driver and use original functoin of Ethernet card, we enabled to set IPG from 8 bytes to 1023 bytes in increments of 1 byte, i.e. 8 ns. Note that for IPG tuning, we don't need any interrupt, so, system overhead is zero.

Figure 7 is the graph of average, standard deviation, maximum and minimum transfer rate for IPG from 8 to 1023 with 64 bytes steps. This graph shows the tendency that when IPG is small, the transfer rate is very different, and as the IPG becomes larger, transfer rate becomes more stable. And, surprisingly, when IPG is smaller than 320 bytes, the worst transfer rate is always less than half of Fast Ethernet in 8 times trial. This may mean, currently, buffer of intermediate routers are not enough to absorb the bursty behavior of Gigabit Ethernet.

It's true that IPG tuning is effective to make bursty behavior of GbE milder, i.e., busy Δt becomes $(IPG+F)/F$ times longer. But, suppose IPG is set to the maximum value, 1023 bytes and packet-level rate Y is 1 Gbps, $FY/(IPG+F)$ is about 600 Mbps, which is faster than flow-level rate X in most cases. This means, there still exist busy Δt and $RTT - \Delta t$ idle time of transmitter.

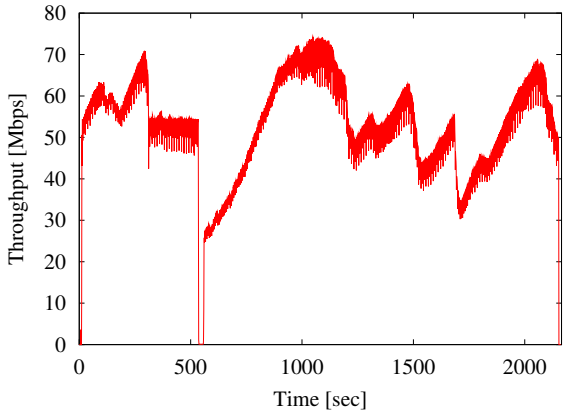

Figure 8: Throughput of on-disk file transfer with 8B IPG

Fig. 8 and 9 shows the throughput of on-disk file transfer with 8-byte and 1023-byte IPG respectively. Extending IPG stabilizes the network behavior and the throughput sustains at the transfer rate of a disk, 453 Mbps on average. Meanwhile, the file transfer with miminum IPG suffers from packet losses and the throughput never exceeds 100 Mbps and 53 Mbps on average.

Fig. 10 – 15 shows the throughput of file transfer with 1023-byte IPG. Each node makes 4 streams, each of which corresponds with one HDD. Fig. 10, 11 shows how 4 nodes share OC-48. Fig. 12, 13, 14, 15 shows how 2 nodes share GbE. Nodes on OC-48 attain about 465 Mbps and nodes on GbE attain from 130 Mbps to 200 Mbps.

But the shortcoming of IPG tuning is that it affectes all communication over LFN-tuned interface.

3 Clustered Packet Spacing

To realize packet spacing by software, we modify the packet transmission scheduling during the slow start to disperse one window of packets over one RTT. Since TCP congestion control is ACK-arrival event-driven and has "self-clocking" nature but the network can't help pacing for high bandwidth-

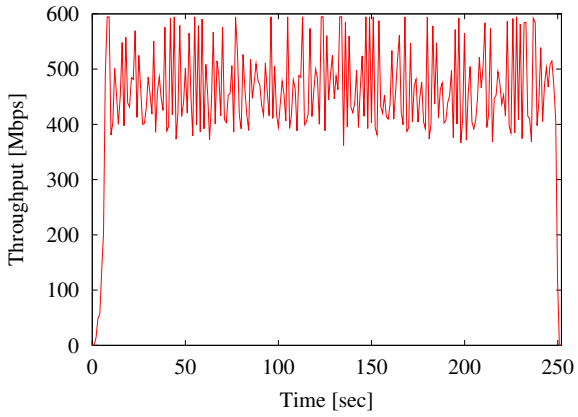

Figure 9: Throughput of on-disk file transfer with 1023B IPG

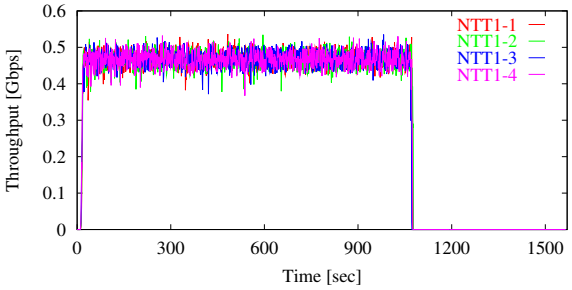

Figure 10: Throughput of file transfer with 1023B IPG. 4 nodes share first OC-48. Each node makes 4 streams

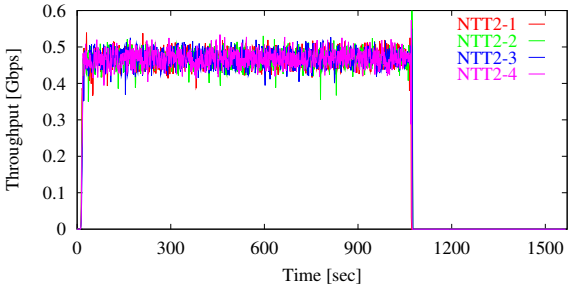

Figure 11: Throughput of file transfer with 1023B IPG. 4 nodes share second OC-48. Each node makes 4 streams

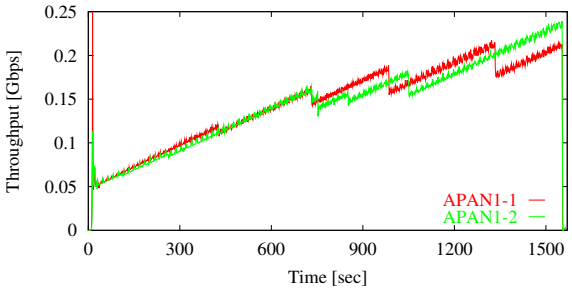

Figure 12: Throughput of file transfer with 1023B IPG. 2 nodes share first GbE for APAN. Each node makes 4 streams

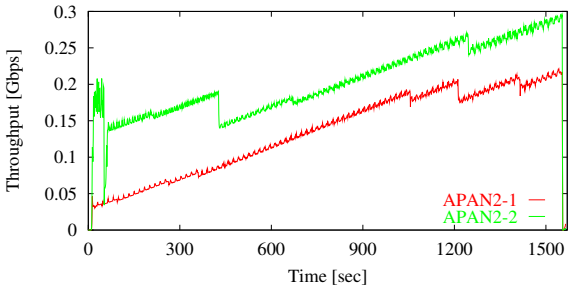

Figure 13: Throughput of file transfer with 1023B IPG. 2 nodes share second GbE for APAN. Each node makes 4 streams

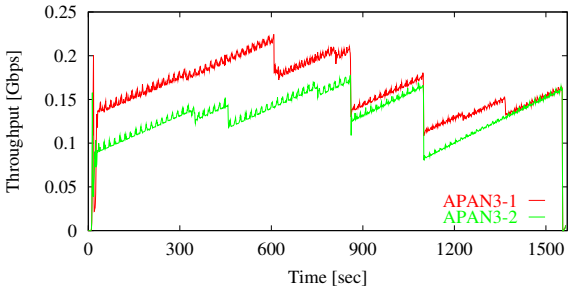

Figure 14: Throughput of file transfer with 1023B IPG. 2 nodes share third GbE for APAN. Each node makes 4 streams

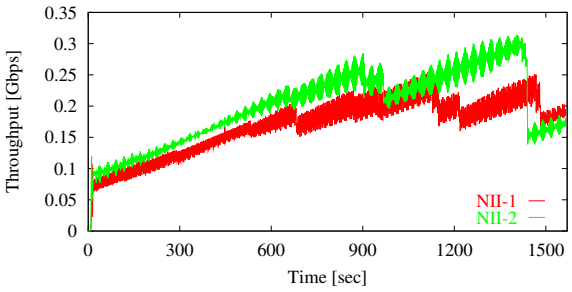

Figure 15: Throughput of file transfer with 1023B IPG. 2 nodes share GbE for APAN. Each node makes 4 streams

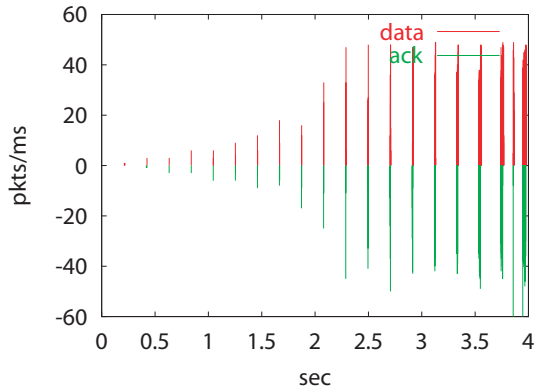

Figure 16: Packet number for first 20 RTTs, RTT 200 msec, first 4 sec

delay-product flows, it's important to distribute packets over one RTT until cwnd grows to appropriate size. After that, the normal congestion control algorithm takes over and these isolated packets are growing into small bursts, which have more chance to be handled without loss in routers, and become one large window as a whole. We can avoid the overhead of deep fine-grained spacing and benefit from the efficiency of the burstiness.

We implement this method using kernel timer in Linux kernel. First, we compute the desired transmission interval $cwnd_{n+1}/RTT$ every RTT where $cwnd_{n+1}$ is the target cwnd during next RTT. Since cwnd is doubled every RTT during slow start, if the initial cwnd is 2 and RTT is 200 ms, then the target interval is 100 ms, 50 ms, 25 ms, and so on.

Figure 16 and Figure 17 show the graph of packet number, which is sent or recieved packets per millisecond, without

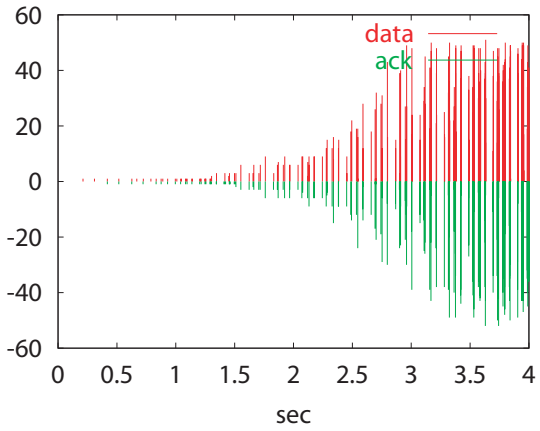

Figure 17: Packet number with Clustered packet spacing

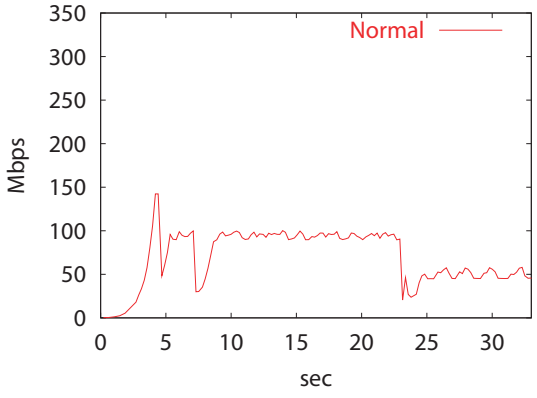

Figure 18: Throughput without Packet Spacing

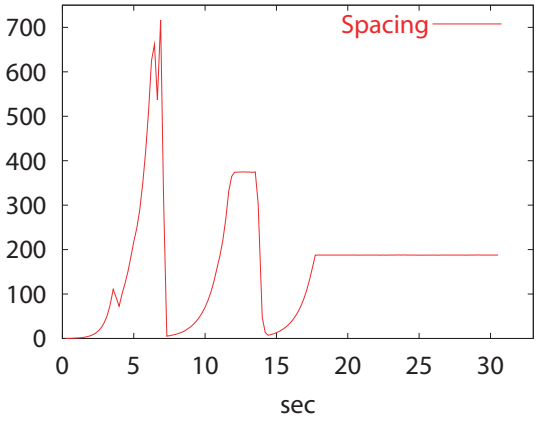

Figure 19: Throughput with Clustered packet spacing

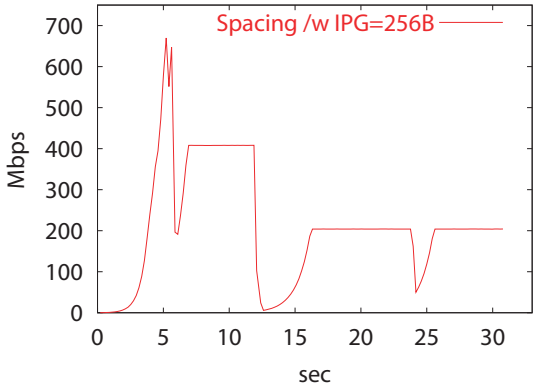

Figure 20: Throughput with IPG 256 bytes with Slow Start Phase Modification

and with Clustered Packet Spacing, where positive number shows the number of sending packets and negative number shows number of received ACKs for every millisecond of first 2 seconds. The former presents ordinary slow start phass and the latter presents modified slow start phase, where packet is distributed for first 1 sec, and after that, number of packets grows. Figure 18 and Figure 19 show the best throughput for the continuous 5 times trial of "without" and "with" modification. Without modification 65.4 Mbps was the best, and with modification 123 Mbps.

4 Combination of IPG tuning and Clustered Packet Spacing

We combine IPG tuning and Clustered Packet Spacing. IPG tuning is fine-grain work such as 1 μsec, (8 nsec per unit up to 1023 $\times$ 8 nsec, and currently we use 256 units), and this is done by MAC layer, and IPG tuning works as the scrape of the peak of the mountain. On the other hand, modification of Slow Start Phase is rather coarse grain work such as 1 msec, and this modification works as the reclamation of the valley. In addition, both method have very little overhead to the system.

Figure 20 shows the best throughput data which we got by 5 times trial. Total throughput is 208 Mbps.

To control transmission rate for multiple streams simultaneously, we conduct packet spacing during slow start phase in TCP stack as follows,

(1) While $RTT/cwnd > T$, send packets with $RTT/cwnd$ interval

(2) If $RTT/cwnd < T$, normal TCP congestion control takes over.

where T is the thresold interval; this is currently the precision of kernel timer, i.e. 10 msec if Linux 2.4 and 1 msec if Linux 2.6.

(1) is the phase of interval control which settles the seed of cluster, and, in (2), i.e., ordinary TCP procedure, each cluster grows individually by each ACKs. The overhead is almost negligible for interval control lasts at most 10 times since the maximum RTT on the earth is about 500 msec of satelite communication.

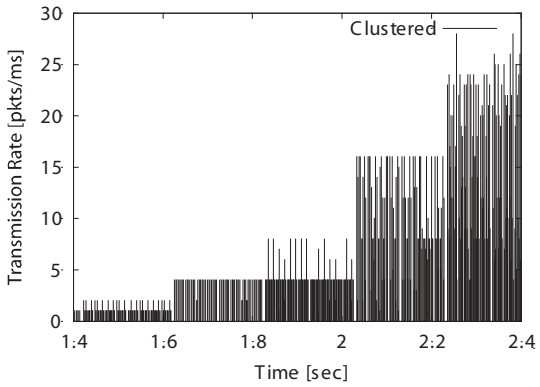

Figure 21: Number of packets for millisecond with Clusterd Packet Spacing, (precision 1 ms, Linux 2.6)

Figure 21 shows the state just after interval control phase

finishes. The effect of Clustered Packet Spacing is that busy Δt is divided into RTT/T pieces and placed every T interval instead of RTT. Then, the load of intermediate buffer is almost same as the data transfer rate when we consider the interval whose granurality is longer than T precision. But since there exist RTT/T peaks, it easily suffers the effects of packet loss when window size grows too large.

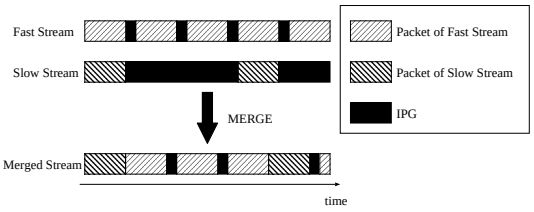

Figure 22: Merge Multi Stream Packets

5 Network Interface Card

5.1 Overview

Since IPG tuning is proceeded in Link layer, it effects all its connections. IPG tuning is very effective for the system whose main purpose is point to point data transfer between fixed places, for example, a scientific project shares huge amount of observed data between observatory and universities. But IPG tuning is not appropriate for a server, which serves huge amount of data to many clients of various RTTs at a time. On the other hand, Clustered Packet Spacing is proceeded in transport layer, which inserts packet space with the assumption that it is possible to estimate the appropriate interval between packets in slow start phase. This assumption is not always true by the network condition. In addition, once the spacing becomes not even by several packet losses, there is no way to recover in Congestion Avoidance Phase. For we dare not to put spacing in Congestion Avoidance Phase to avoid expected overheads. The problems of both IPG tuning and Clustered Packet Spacing are that it uses only information of its layer. Here, let us recall packet interval; Gigabit Ethernet is about 12.5μseconds, and 10Gbit Ethernet is about 1.3 μ seconds for each ethernet frame. It is almost impossible to adjust this spacing all the time using main CPU with negligible overhead. Hence, we design TCP-aware ethernet network interface card. The main purpose of this TCP-aware NIC is to recognize each stream and get information of its window size and RTT, then, merge these streams into series of packets whose packet interval is appropriately adjusted and no stavation occurs. Figure 22 shows the example of two TCP streams are merged into a sequence of ethernet packets.

5.2 Architecture

Figure 23 shows main functional blocks. XLINX PCI Core issues PCI transaction via PCI Interface. When a packet comes, we find its connection ID by using CAM. The address of SRAM can be calculated from the ID. Packets from host are put into Send Buffer and from network are put into

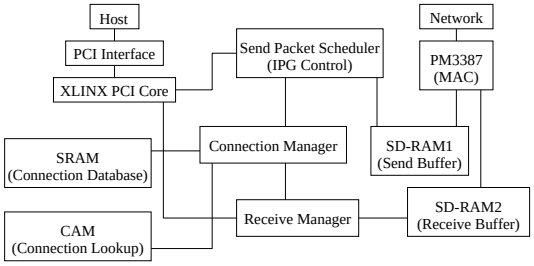

Figure 23: Overview of Our NIC Architecture

Receive Buffer. The data stored in Send Buffer will be scheduled to be sent by Send Packet Scheduler. The scheduled data is transmitted to network via MAC (PM3387) Received data will be passed to host via PCI blocks by Receive Manager.

The send packet procedure on our NIC is as follows:
(1)A packet comes from host via PCI.
(2)Put it into the Send Buffer.
(3)Connection Manager accesses CAM and find a connection ID.
(4)Calculate a SRAM address.
(5)Update pointers in the SRAM.
(6)Wait until scheduled.
(7)Send to network via PM3387 (MAC).
(8)Get a new IPG value from SRAM.
(9)Set the value to the IPG Counter in Send Packet Scheduler.
We apply the linked list structure to the Send Buffer, because normal FIFO buffer can't be applied, since the buffer size which will be requested by each stream is difficult to be expected. A pointer is added to the head of each packet data. This points next packet of same stream. The head and tail packets of each stream's linked list are pointed from SRAM. When stream i is scheduled, the packet pointed by head pointer from SRAM will be transmitted next. After the transmission, the area of the packet will become NULL state and ignored. When a new packet comes, it is simply put into

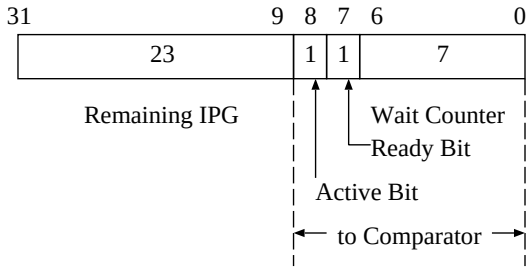

Figure 24: Structure of 32 bit IPG Counter

the next of latest packet like a ring buffer and pointers are updated. If Send Buffer has enough size, the reuse of NULL state area doesn't need to be considered until the pointer to a latest packet comes back after 1 round. Because, the maximum IPG time of our NIC is about 536 ms, all packets send at least in 1 second. SRAM has the following information of each connection.
(1) RTT
(2) IPG value
(3) pointer to the head of the stream in the Send Buffer
(4) pointer to the tail of the stream in the Send Buffer
The correspondence between each stream and its SRAM address is searched by Connection Manager. It looks up the CAM by using unique stream information and find quickly the connection ID. After that the SRAM address will be calculated from the ID. The Packet Send Scheduler consists of IPG Counters and Comparators. A 32 bit IPG counter is assigned to each stream. The counter consists of 4 parts as Figure 24

When a packet transmission finished, a new IPG value is set to the Remaining IPG part. If the Remaining IPG part is n, it means $8 \times n$ byte spacing. In other word, the Remaining IPG is reduced by 1 every 64 ns. The Remaining IPG part has 23 bit width. This means the maximum IPG value is over 500 ms. While connection is active, Active Bit is always fixed to 1. When Remaining IPG part is 0 and transmission data exist in Send Buffer, then Ready Bit becomes 1. If connection isn't active, then Ready Bit must be 0. If two or more streams are ready to send, one of them is chosen and the Wait Counters of lose streams are increased. The bit 8 to 0 of the counter is connected to a comparator. The Comparators used by the Packet Send Scheduler have 9 bit $\times$ 2 input ports. It compares two 9 bit values and returns bigger one. By using the combination of IPG Counters and Comparators, the packets buffered in Send Buffer will be merged as we wish.

Packet exchanges between host OS and NIC are via PCI bus with DMA. When a DMA transfer finished, then interruption will be occurred and Host OS can notice the completion. To access SRAM we need to access to CAM. To access CAM we need to know source IP address, source port number, destination IP address and destination port number.

6 Simulation

We made two types of simulations. One is a performance simulation on single stream. We measure throughput and standard deviations under various conditions. The other is two stream simulation. Fast stream and slow stream send data concurrently, and we confirm our Packet Send Scheduler works correctly.

6.1 Simulation Environment

The simulation is done on SunFire 15000. It has 72 $\times$ 900 MHz UltraSPARCⅢ processors and 288 GB Memory. OS is Solaris8 and compilers are "cc" of Sun WorkShop 6 and "vhdlan" of Synopsys. The compiler options of "cc" are "-xarch=v9 -xO4". Our simulator can simulate the arbitrary structure of networks. Each host's bandwidth can be changed freely up to 1G bps. The bandwidth and latency of each network can also be changed. The maximum bandwidth of networks is 1G bps and no limitation to latency.

We think networks as large buffers and packet drops are determined in the networks. The buffer size of a network is product of bandwidth $\times$ latency. Basic packet loss rate in our simulator is $\frac{1}{100000}$.

When burst communication is done, packet loss rate is inflated. To detect the burstiness, we introduce "BurstCount". When a packet comes, its size is added to BurstCount. And BurstCount is decreased to half every 1 ms. The actual packet loss rate is calculated as follows:

$$\mathrm{P} \quad = \quad \left(1 + \frac{\text{BurstCount}}{\text{Buffer Size}}\right) \times \frac{1}{100000}$$

RED (Random Early Detection) is also applied. The RED threshold changes with latency. When the latency is under 3 ms, it is 0.99 and when latency is over 55 ms, it is 0.50. The measured data between Tokyo and Baltimore is referenced to determine these simulation parameters. The simulator is written in C and VHDL. As the interface between C and VHDL, we use VHPI (VHDL Programming Interface).

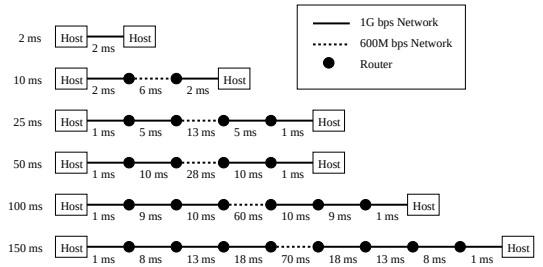

Figure 25: Network Configuration on Single Stream Simulation

6.2 Single Stream Simulation

The bandwidth of hosts are 1Gbps or 100Mbps in the simulation. There are two types of IPG configurations. One is the **Dynamic IPG Mode**. In this mode, each host changes their IPG dynamically. Another is the **Fixed IPG Mode**. In this mode, IPG is fixed regardless its window size and latency. The IPGs tested in the Fixed Mode simulation is as follows:
8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4196, 8192, 16384 bytes
In the 100M bps mode, IPG is fixed to 8 byte, because communications on 100M bps is originally not bursty. In our simulation, following latencies[1] are tested.
2 ms, 10 ns, 25 ns, 50 ns, 100 ns, 150 ns

Fundamentally, the bandwidth of networks are set to 1Gbps. But, each configuration has a bottleneck line without 2ms latency case. The bandwidth of the bottleneck line is 600M bps. The detail configuration is shown in Figure 25.

In each combination of host and network, 300MB data transmission is performed. We measure their throughput and standard deviations. Simulation is performed 10 times for each parameter. Figure 26 shows the throughput of Gigabit Ethernet and Fast Ethernet. Both of them use default IPG (8 byte). In Figure 26, x-axis means 1-way latency and y-axis means throughput.

Note that Fast Ethernet is faster than Gigabit Ethernet when latency is over 100 ms[20]. Figure 27 shows the normalized standard deviation of Gigabit Ethernet and Fast Ethernet. Both of them use default IPG (8 byte). In Figure 27, x-axis means 1-way latency and y-axis means normalized standard deviation.

[1] 1-way latency

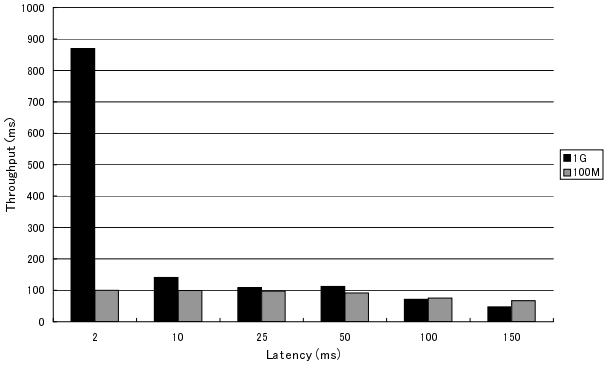

Figure 26: Throughput of Gigabit and Fast Ethernet

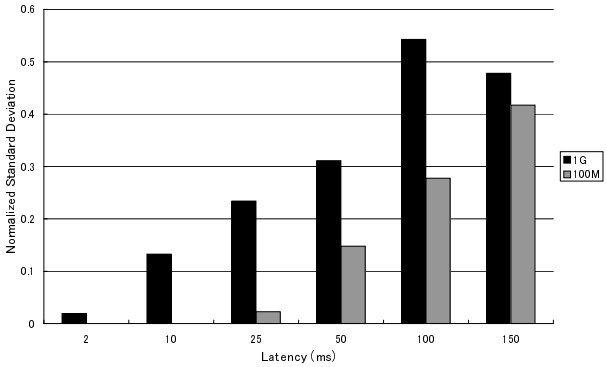

Figure 27: Normalized Standard Deviation of Gigabit and Fast Ethernet

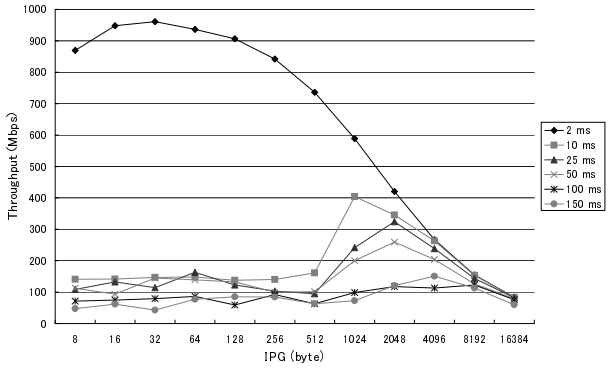

Figure 28: Throughput of Gigabit Ethernet on Various IPG

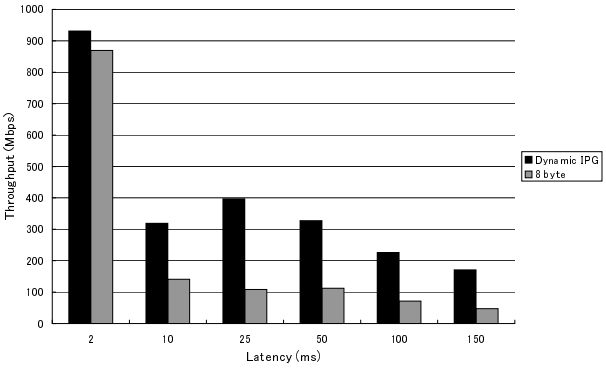

Figure 30: Throughput Comparison between Dynamic IPG and Default IPG

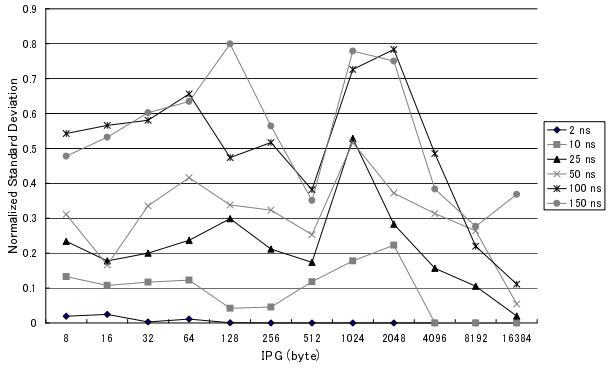

Figure 29: Normalized Standard Deviation on Various IPG

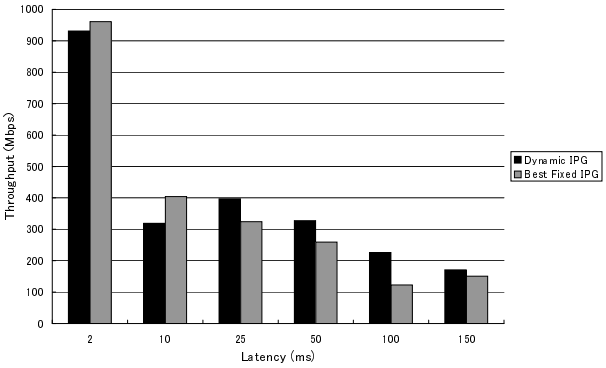

Figure 31: Throughput Comparison between Dynamic IPG and Best Fixed IPG

Fast Ethernet is more stable than Gigabit Ethernet in all cases. Figure 28 shows the throughput of Fixed IPG mode. In figure 28, x-axis means IPG and y-axis means throughput.

The IPG which achieves peak performance tends to be large as the network delay becomes longer. However, too big IPG cause serious performance degradation like 16384 byte IPG. Figure 29 shows the normalized standard deviations of Fixed IPG mode. In figure 29, x-axis means IPG and y-axis means normalized standard deviation.

The normalized standard deviation tends to be large in long latency and it becomes small when IPG size is enough large. The throughput of Dynamic IPG and Default IPG transmissions are shown in Figure 30. In Figure 30, x-axis means 1-way latency and y-axis means throughput.

In all cases, Dynamic IPG mode achieve higher performance than Default IPG mode. Figure 31, compares the throughput of Dynamic IPG and the best case of Fixed IPG. In Figure 31, x-axis means 1-way latency and y-axis means throughput.

When latency is over 25 ms, Dynamic IPG mode achieves higher performance than the best case of Fixed IPG mode. When latency is small, the best case of Fixed IPG mode is slightly better. The normalized standard deviations of Dynamic IPG and Default IPG transmissions are shown in Figure 32.

In all cases, Dynamic IPG mode are more stable than Default IPG. Figure 33, compares the normalized standard deviation of Dynamic IPG and the fixed IPG which achieves best throughput. In Figure 33, x-axis means 1-way latency and y-axis means normalized standard deviation.

The normalized standard deviations of fixed IPG which achieve best throughput, are always lager than Dynamic IPG

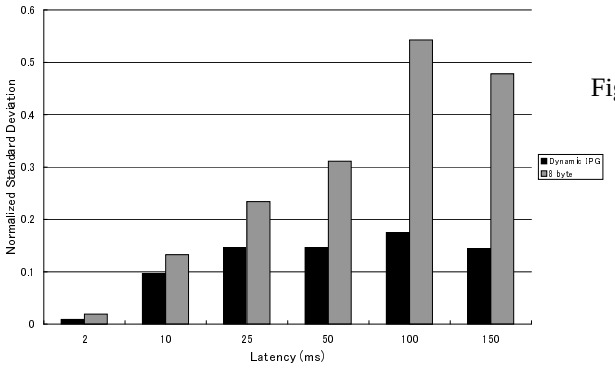

Figure 32: Normalized Standard Deviation of Dynamic IPG and Default IPG

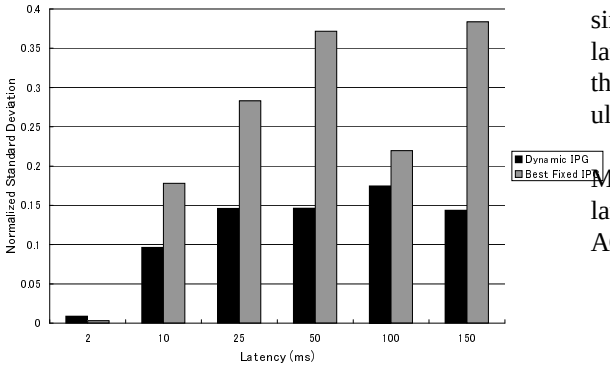

Figure 33: Normalized Standard Deviation of Dynamic IPG and Best Fixed IPG

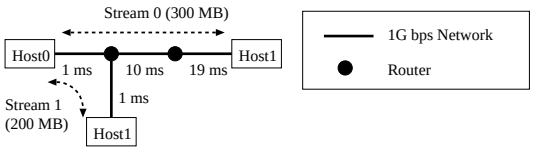

Figure 34: Network Configuration on Two Stream Simulation

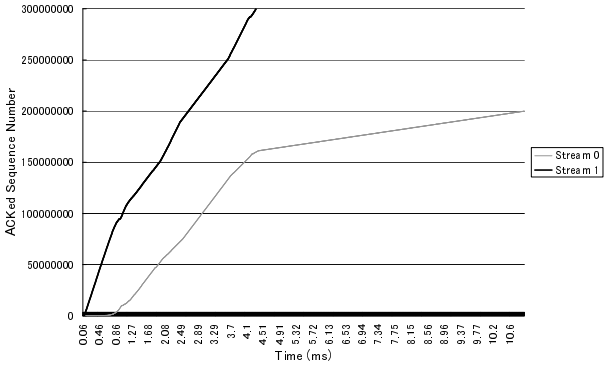

Figure 35: Data Transmission of Two Stream

mode.

6.3 Two Stream Simulation

To check scheduling ability of Packet Send Scheduler, we simulate the two stream transmission. Stream 0 has 30 ns latency and Stream 1 has 2 ns latency. They shares part of their path as shown in Figure 34. All of networks in this simulation have 1G bps bandwidth.

Stream 0 sends 200 MB of data and Stream 1 sends 300 MB of data. Figure 35 is the result of the two stream simulation. In Figure 35, x-axis means time and y-axis means the ACKed sequence number at that time.

The throughput of stream 0 and 1 are as follows:

	Stream 0	Stream 1
Throughput	146.92 Mbps	561.64 Mbps

Stream 1 sends more data than stream 0, but starvation doesn't occur. This means scheduler works correctly.

7 Related Works

Packet spacing has been well known method to stabilize Internet traffic. The closely related ancestor of packet spacing

is rate-based control of network traffic [1], but actual behavior and effects of controlling intervals of packets has not been investigated until recently [19]. In [19], authors shows positive effect of packet spacing to reduce packet loss of UDP streams on actual Internet.

Application of packet-level rate control to TCP has been studied for improving efficiency degradation due to bursty nature of traffic caused by TCP congestion control mechanism. [2] performed detailed simulation on LFN. However, their observations are not consistent with our observation described in this paper. According to the result of [2], controlling packet intervals mainly has negative effect to increase throughput. But we have observed positive effect of packet spacing on TCP traffic. This difference comes from the selection of network parameters. Since we use real network circuit, the bottleneck throughput is 10-20 times faster, and RTT is twice longer and many routers exist between end nodes. Though the interval of saw-tooth pattern of TCP window size is proportional to available bandwidth and RTT, but we cannot see saw-tooth pattern and the peak is decreasing as time elapses. One reason of the difference could be that the ratio of routers' buffer size against TCP window size is smaller than [2].

HighSpeed TCP (HSTCP) [13], Scalable TCP (STCP) [14] and FAST TCP [15] try to change the window size more aggressively than standard TCP. HSTCP modifies AIMD parameter to a function of current window size. STCP uses MI (Multiplicative Increase) instead of AI. Both of them accelerate the growth of window as the window grows.

While HSTCP and STCP use only packet loss as a congestion signal, FAST TCP uses both packet loss and RTT variance, like TCP Vegas [12]. FAST TCP considers a queueing delay causes RTT variance and decelerates the AI according to the amount of queueing delay.

Other modifications [16, 17] try to make the convergece of competing streams faster in addition to accelerating AI.

We focus on how we can reduce packet losses and how we can makes occurences of packet loss fair among competing streams. Then we adopt a packet spacing instead of modification of congestion control algorithm.

On the other hand, some theoretical analyses [4, 5, 6, 7] have been published recently. They are *competitive* analyses in some limited environments, such as limited network topology, limited model of loss probability and of router's buffer, and without propagation delay of congestion signal.

8 Concluding Remarks

We claim that window size control is not the only problem of LFN to be solved. The difference of flow-level rate and packet-level rate causes needless overload to intermediate routers on the path and, will definitely cause more serious problem when 10Gbps Ethernet is commonly used.

We show two software approaches of adjusting packet-level rate to approximate flow-level rate; one is proceeded in link layer, and the other is proceeded in transport layer. Experiments on real LFN show that these approaches are effective, and combination of these approaches are also effective. As for future works, we want to check the combination of our software approaches and HighSpeedTCP, FastTCP, and ScalableTCP. Slow down at appropriate moment in Slow Start Phase is our another target.

Finally, we believe that when 10Gbps era comes, NIC which can recognize TCP-stream, in other words, cooperation and information sharing between link layer and transport layer, which is still under development and only simulation result is shown in this paper, shall be very helpful.

Acknowledgement

Special thanks to Prof. Akira Kato of University of Tokyo for useful discussions, arranging long distance experiments, and pushing me to submit this paper to PFLDnet. This study is partially supported by the Special Coordination Fund for Promoting Science and Technology, and Grant-in-Aid for Fundamental Scientific Research B(2) #13480077 from Ministry of Education, Culture, Sports, Science and Technology Japan, Semiconductor Technology Academic Research Center (STARC) Japan, CREST project of Japan Science and Technology Corporation, and by 21st century Center Of Excellence project of Japan Society for the Promotion of Science. The data transfer experiment is supported by APAN and Wide Project. We thank Mr. Kamezawa, Mr. Tamadukuri and Ms. Aoshima of University of Tokyo, Mr. Kurusu, Mr. Ikuta, Mr. Sakamoto, Mr. Furukawa, Mr. Yanagisawa, Mr. Nakano, Mr. Torii, and Mr. Mizuguchi of Fujitsu Computer Technologies, and Dr. Matsuo, Dr. Masuoka, Mr. Zinzaki, and Mr. Shitami of Fujitsu Laboratory, and Mr. Miura and Mr. Sudo of Digital Technology.

References

[1] D. Clark, M. Lambert, L. Zhang, ”NETBLT: A Bulk Data Transfer Protocol”, RFC 998, Mar. 1987.

[2] A. Aggarwal, S. Savage, T. Anderson, “Understanding the Performance of TCP Pacing”, INFOCOM 2000, Mar. 2000.

[3] D. M. Chiu and R. Jain, ”Analysis on the increase and decrease algorithms for congestion avoidance in computer networks”. Computer Networks and ISDN systems, 17(1), June 1989.

[4] R. Karp, E. Koutsoupias, C. Papadimitriou, S. Shenker, ”Optimization problems in congestion control”, 41th FOCS, pp.66-74, Nov. 2000.

[5] J. Edmonds, S. Datta, P. Dymond, ”TCP is competitive against a limited adversary”, SPAA 2003, pp.174-183, Jun. 2003.

[6] T. Itoh and M. Inaba, ”Theoretical Analysis of Throughput of TCP/IP Congestion Control Algorithm with Different Distances”, IPSJ SIG Notes SIGAL, Jan. 2004.

[7] Tsuyoshi Itoh and Mary Inaba, ”Theoretical Analysis of Performances of TCP/IP Congestion Control Algorithm with Different Distances”, Networking 2004, May 2004, To appear.

[8] T. Hacker, B. Athey, and B. Noble, ”The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network”, Proc. 16th Int’l Parallel and Distributed Processing Symposium (IPDPS), 2001.

[9] T. Hacker, B. Noble and B. Athey, ”The Effects of Systemic Packet Loss on Aggregate TCP Flows”, SC2002, Nov. 2002, http://www.sc-2002.org/paperpdfs/pap.pap270.pdf

[10] V. Jacobson, ”Congestion Avoidance and Control”, Proc. ACM SIGCOMM ’88, pp.314-332, Aug. 1988.

[11] J. C. Hoe, ”Improving the Start-up Behavior of a Congestion Control Scheme for TCP”, Proc. ACM SIGCOMM ’96, pp.270-280, Aug. 1996.

[12] L. Brakmo and L. Peterson, ”TCP Vegas: End to End Congestion Avoidance on a Global Internet”, IEEE Journal of Selected Areas in Communications, 13(8):1465-1480, October 1995.

[13] S. Floyd, ”HighSpeed TCP for Large Congestion Windows”, RFC 3649, Dec. 2003.

[14] T. Kelly, ”Scalable TCP: Improving Performance in Highspeed Wide Area Networks”, PFLDnet 2003.

[15] C. Jin, D. Wei, and S. Low ”FAST TCP: Motivation, Architecture, Algorithms, Performance”, Infocom 2004, Mar. 2004. To appear.

[16] R.Shorten, D.Leith, J.Foy, R.Kilduff, ”Analysis and design of congestion control in synchronised communication networks”

[17] L. Xu, K. Harfoush, I. Rhee, ”Binary Increase Congestion Control for Fast Long-Distance Networks”, Infocom 2004 Mar. 2004. To appear.

[18] J. Mahdavi, and S. Floyd, ”TCP-Friendly Unicast Rate-Based Flow Control”, Technical note sent to the end2end-interest mailing list, Jan. 1997.

[19] T. Miyata, H. Fukuda, and S. Ono, ”Impact of Packet Spacing Time on Packet Loss under LW for QoS of FEC-based Applications”, IPSJ Sig Notes, Mobile Computing, Vol.6, pp.7-13, 1998 .

[20] M. Nakamura, M. Inaba, and K. Hiraki, ”Fast Ethernet is sometimes faster than Gigabit Ethernet on Long Fat Pipe Network environment - Observation of congestion control of TCP streams”, PDCS2003, Nov. 2003.

[21] K. Hiraki, M. Inaba, J. Tamatsukuri, R. Kurusu, Y. Ikuta, H. Koga, and A. Zinzaki, ”Data Reservoir: Utilization of Multi-Gigabit Backbone Network for Data-Intensive Research”, SC2002, Nov. 2002. http://www.sc-2002.org/paperpdfs/pap.pap327.pdf

[22] S. Nakano, and et al. ”DR Giga Analyzer”, Symp. on Global Dependable Information Infrastructure, Feb. 2004. (in Japanese)

[23] S. Mukherjee and M. Hill, ”The Impact of Data Transfer and Buffering Alternatives on Network Interface Design”, Proc. 4th Int’l Symp. on High-Performance Computer Architecture (HPCA), Feb. 1998.

[24] H. Sivakumar, S. Bailey and R. Grossman, ”PSockets: The case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks”, SC2000, Nov. 2000.

[25] Lars Eggert, John Heidemann, and Joe Touch, ”Effects of Ensemble-TCP”, ACM Computer Communication Review, 30 (1), pp.15-29, Jan. 2000.

[26] T. Hacker, B. Noble and B. Athey, ”Improving Throughput and Maintaining Fairness using Parallel TCP”, IEEE Infocom 2004, Mar. 2004.

[27] : BBftp.
http://doc.in2p3.fr/bbftp/.

[28] : pftp.
http://www.indiana.edu/ rats/research/hsi/.

[29] : GridFTP.
http://www.globus.org/datagrid/.