

Inter-layer coordination for parallel TCP streams on Long Fat pipe Networks

Hiroyuki Kamezawa, Makoto Nakamura, Junji Tamatsukuri,
Nao Aoshima, Mary Inaba, and Kei Hiraki
University of Tokyo
7-3-1 Hongo Bunkyo-ku Tokyo Japan 113-0033
{kame,makoto,junji,aoshima,mary,hiraki}@is.s.u-tokyo.ac.jp

Junichiro Shitami and Akira Jinzaki
Fujitsu Laboratories
{shitami, zinzin}@flab.fujitsu.co.jp

Ryutaro Kurusu, Masakazu Sakamoto,
and Yukichi Ikuta
Fujitsu Computer Technologies
{ryu, msakam, ikuta}@ctec.fujitsu.com

ABSTRACT

As the network speed grows, inter-layer coordination becomes more important. This paper shows 3 inter-layer coordination methods; (1) “Comet-TCP”; cooperation of data-link layer and transport layer using hardware, (2) “Transmission Rate Controlled TCP (TRC-TCP)”; cooperation of data-link layer and transport layer using software, and (3) “Dulling Edges of Cooperative Parallel streams (DECP)”; cooperation of transport layer and application layer. We show the experimental results of file transfer at Bandwidth Challenge in SC2003; one and a half round trip from Japan to U.S., 15,000 miles, which has 350 ms RTT and 8.2 Gbps bandwidth. Comet-TCP hardware solution attained max 7.56 Gbps using a pair of 16 IA servers, which is 92% of available bandwidth and DECP software attained max 7.01 Gbps using a pair of 32 IA servers.

1. INTRODUCTION

1.1 Inter-layer coordination

With the rapid progress of network technology, such as optical fiber and network switches, speed of network grows faster than that of CPU, memory, or other computer devices. Coordination over layers becomes more and more important to attain actual high performance on high speed network. TCP offload interface card is now shipped, for preventing

redundant memory copy or heavy-burden interruption of main CPU per packet. Intrusion detection or encryption and decryption using ethernet card is also helpful to invent efficient system, which we call “inter-layer coordination”. As the speed of network grows, network-specific function is going to be cast into hardware at the end-nodes[18], just like graphics engine on video card computes instead of main CPU.

This paper tackles the instability problem of streams mainly caused by the difference of the rates specified by data-link layer and transport layer.

1.2 Data Reservoir System

“Data Reservoir” system is a file sharing facility to support data intensive scientific research projects, whose sites are spread in distant locations but connected each other by high-speed network. Data Reservoir system consists of several file servers and many disk servers. We adopt low level protocol iSCSI (internet SCSI) for accessing disks and for transferring data. For local access, a disk server plays the role of a *target* like an ordinary hard disk. For bulk data transfer from site *A* to site *B*, a disk server of site *A* plays the role of an *initiator* and a disk server of site *B* plays the role of a *target*. We use parallel streams of striped data in low level to avoid overheads due to disk seek operations context-switch overhead and file system overhead (Figure 1). One of the biggest features of Data Reservoir system is file system transparency achieved by data sharing at storage device level; which enables users to use any existing software without even compilation. Detailed description of Data Reservoir system is found in [7].

1.3 TCP/IP on Long Fat Pipe Network

TCP/IP is a standard end-to-end protocol for reliable transfer. It is well known that TCP has a limitation on performance when it is used for data transfer over long distance and high bandwidth network, called “Long Fat pipe

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SC2004 November 6-12, 2004, Pittsburgh, PA, USA
0-7695-2153-3/04 \$20.00 (c)2004 IEEE.

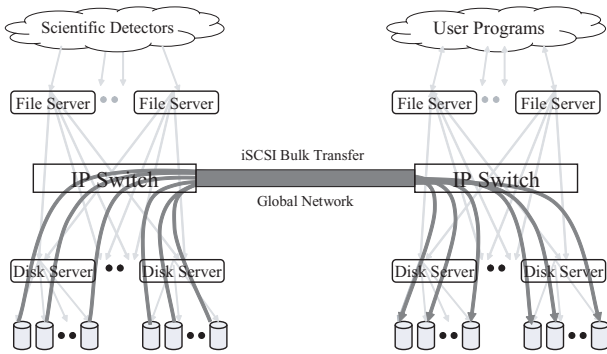


Figure 1: Burst data transfer

Network (LFN)". For reliability, TCP uses ACK; a sender keeps data for re-transmission until ACK returns. Data, which is sent but not ACKed, is called "inflight" data, and its maximum size is called "window size". Data transfer rate of TCP is roughly $window\ size / RTT$, where RTT is Round Trip Time. Long distance data transfer needs large window size, since transfer rate is inverse proportional to RTT . In addition, speed of growth of the window size is proportional to RTT , since it takes RTT time till ACK returns. The window size is controlled by widely used loss based TCP algorithms as follows; in *slow start phase* doubled for every RTT , and in *congestion avoidance phase* "Additive Increase and Multiplicative Decrease(AIMD)" manner, i.e., linearly increased while transfer succeeds, and halved by transfer failure. Many proposals have been shown to change the window size control algorithm; such as Scalable TCP (STCP) [12], HighSpeed TCP (HSTCP) [4] and FAST TCP [9].

But the problem of bad performance on LFN is not only because of the slow growth of window size. For example, performance of streams in same condition disperse a lot, which is a serious problem when system use parallel streams because the slowest stream is the dominant factor. In our experiment, dispersion is observed only when we use Gigabit Ethernet interface at the end-point, where bottleneck is OC-12. And, when we change interface to FastEthernet, this dispersion disappears immediately, although TCP stack is completely same [14]. We guess this dispersion may be caused by shortage of buffers of intermediate routers; that is, although macroscopically there exist no congestion, microscopically there exist buffer overflow in intermediate routers, which occasionally induces unlucky packet loss, which results in unnecessary dispersion of performance. This problem becomes more serious when we use 10 Gbps interface.

1.4 Overview

We propose 3 inter-layer coordination methods; (1) "Comet - TCP", a hardware solution and (2) "Transmission Rate Controlled TCP (TRC - TCP)", a software solution to cooperation of data-link layer and transport layer, and (3) "Dulling Edges of Cooperative Parallel streams (DECP)", cooperation of transport layer and application layer.

In this paper, first, we show our observation of Japan-US data transfer experiment in SC2002 between Tokyo and Baltimore whose bottleneck was OC-12. Then, we explain the problems we found. Next, we show our proposals and

implementations. Finally, we show the experimental results of these proposals in Bandwidth Challenge in SC2003. We transfer data file for one and a half round trip from Japan to U.S., 15,000 miles, which has 350ms RTT and 8.2 Gbps bandwidth. Hardware solution Comet-TCP attained max 7.56 Gbps using a pair of 16 IA servers, and software DECP max 7.01 Gbps using a pair of 32 IA servers.

2. OBSERVATIONS AND PROBLEM

2.1 Experiments over the Pacific Ocean with bottleneck OC-12

In November 2002, we experimented "Data Reservoir" file sharing system. Endpoint sites are University of Tokyo in Tokyo and SC2002 exhibition hall in Baltimore, where distance is 7500 miles, RTT is 200 ms, and APAN¹ OC-12/POS is the bottle neck. In this experiment, we attained 91% us-

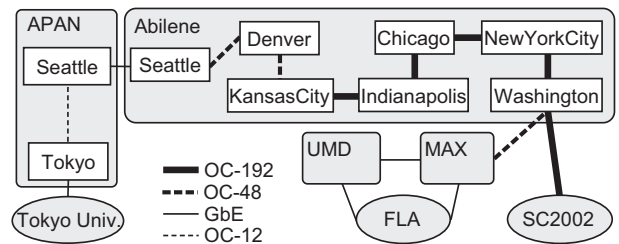


Figure 2: Network between Tokyo and Baltimore

age of bandwidth using a pair of 26 IA servers when we used bottleneck line exclusively (Figure 3). But, we encountered two problems. One is the low performance of each stream; a server can attain at most 20 Mbps in average. The other is the disperse speed of streams. Even though the machine specification is equal, data is equally divided, and each machine takes equal load, sometimes certain threads needs longer time. For example, from 150 sec to 400 sec in Figure 3, the number of active streams decreases little by little, because faster streams finish its job gradually. Figure 4 shows the sequence number of the fastest and the slowest streams out of 4 streams. One thread needs three times longer time to finish its job than the fastest thread. This results in that total performance becomes one third, since the slowest stream is the dominant factor of the system performance. Even after other streams finish their jobs, and even though no competitor exists, slower streams tend to fail to gain that bandwidth.

2.2 Gigabit Ethernet Interface vs. Fast Ethernet Interface

After the Bandwidth Challenge, we experimented data transfer of single stream between FLA (Fujitsu Laboratory America) in Maryland and University of Tokyo (Figure 2), using a pair of IA servers. We compare throughput of PC with interface Gigabit Ethernet (GbE) and Fast Ethernet (FE) using "iperf" software, while tuning TxQueue length. Table 1 shows minimum, maximum, and average transfer rate of GbE and FE. The performance of GbE is not stable at all and varies from 5.6 Mbps to 120 Mbps with TxQueue

¹Asia-Pacific Advanced Network. <http://www.apan.net/>

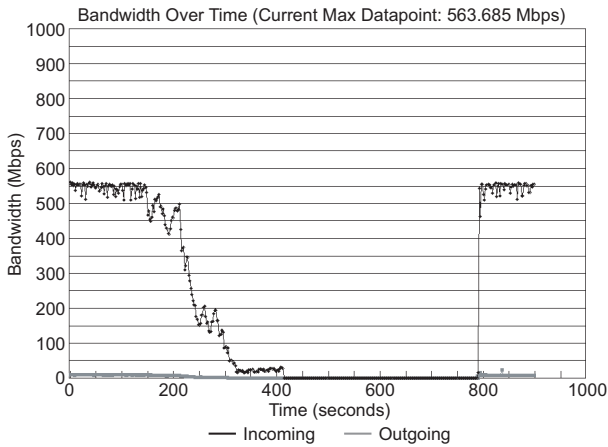


Figure 3: 7500 miles bulk data transfer with exclusive bottleneck usage

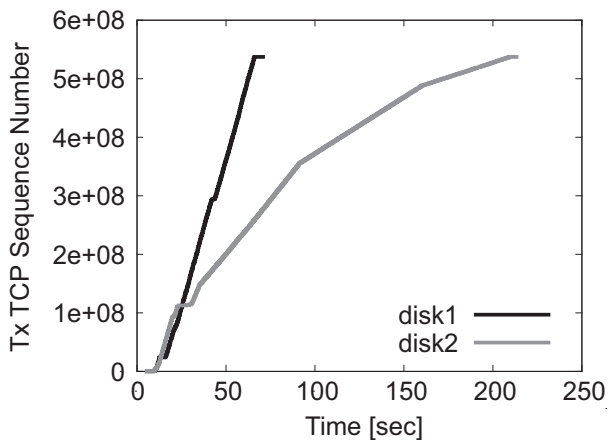


Figure 4: Sequence number (diff) of the fastest and the slowest stream

size 100 and from 9.2 Mbps to 172 Mbps with TxQueue size 25000. On the other hand, FE achieves always about 25.6 Mbps stably with TxQueue size 100, and around 88.7 Mbps with TxQueue size 5000. It was our big surprise that tuned FE is faster than tuned GbE on average and the minimum throughput of GbE is much worse than FE.

So, we observed detailed packet log, using DR-Giga Analyzer [15]. Figure 5 for GbE and Figure 6 for FE show the number of packets received for every millisecond. GbE is peaky and idle time is long. On the other hand, FE sends and receives packet constantly, about 8 packets per ms, and no peak exists. This means, even if the throughputs are same, behavior of packets completely differs according to its network interface.

2.3 Flow-level Rate and Packet-level Rate

Since the size of inflight data is transferred for RTT time, data transfer rate of TCP stream is roughly $window\ size / RTT$. We call this rate as “flow-level rate” of transport layer. But this flow-level rate is too macroscopic when we consider data transfer on LFN with high speed interface such as Gigabit

| | TxQueue | Min | Max | Average |
|---------------|---------|------|-------|---------|
| Fast Ethernet | 100 | 25.6 | 25.6 | 25.6 |
| Ethernet | 5000 | 88.7 | 88.7 | 88.7 |
| GbE | 100 | 5.8 | 120.0 | 45.1 |
| Ethernet | 25000 | 9.2 | 172.0 | 63.9 |

Table 1: Transfer rate in Mbps of Fast Ethernet and Gigabit Ethernet. RTT is 198 msec

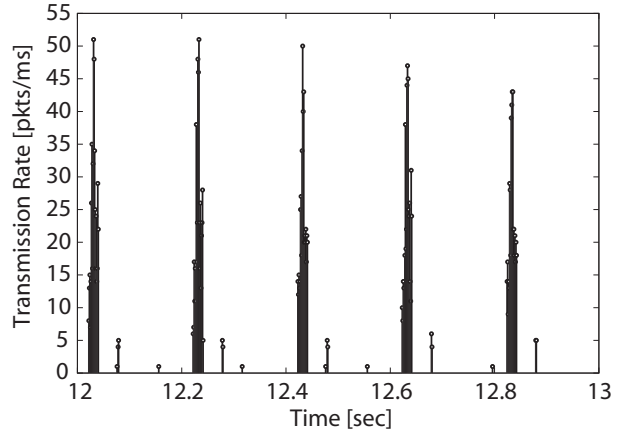


Figure 5: Number of received packets per ms of Gigabit Ethernet, RTT is 198 ms and bottleneck is OC-12

Ethernet (GbE). Suppose one packet is 1500 bytes, packets are transmitted every $12.5 \mu s$ from GbE interface as far as a queue for Ethernet is not empty. We call this microscopic rate as “packet-level rate”.

Let X denote flow-level rate and Y packet-level rate. While transferring data, packets are 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 suffers needless load, $Z = (Y - X) \times \Delta t$, which is often absorbed by buffer of routers. This peaky behavior for busy Δt easily causes needless shortage of buffer memories of intermediate routers which results in miss-leading packet losses to misunderstand that there exists congestion. Note that busy Δt is long when RTT is long, and buffer size Z needs to be large when packet-level rate Y is large and RTT is long. We consider the way how to decrease dispersion of the performance of streams. We try to equalize packet-level rate and flow-level rate, which may decrease the microscopic burden of intermediate routers. We also consider to balance the performance of parallel streams directly by exchanging information and controlling the flow-level rate of parallel streams.

3. COMET-TCP

Comet-TCP is a hardware approach to stabilize data transfer by TCP on LFN. Comet-TCP utilizes large buffer memory on Comet network interface card and hardware QoS capability to send outgoing packets at a constant transmission rate. Figure 7 shows TCP transmission on a pair of Comet-TCP network interface card. No modification is required for application programs on both sides because Comet-TCP has

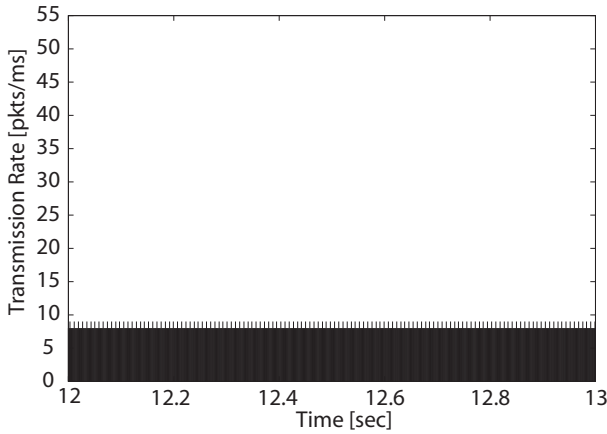


Figure 6: Number of received packets per ms of Fast Ethernet, RTT is 198 ms and bottleneck is OC-12

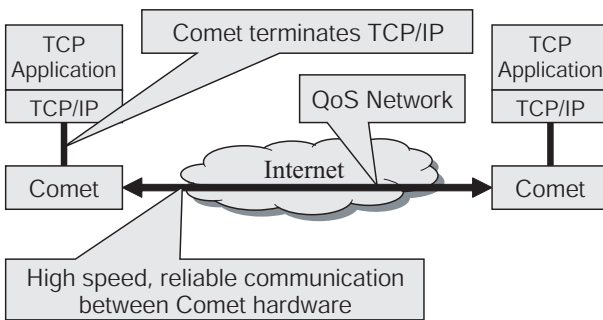


Figure 7: TCP on a pair of Comet-TCP network interface card

compatible API to conventional TCP protocol stack. All the packets that are not ACKed by the remote TCP receiver are stored into packet buffer memory on the local Comet-TCP network interface card (Figure 8). When a sender sends a packet, Comet-TCP immediately terminate TCP by sending back an ACK to a TCP driver unless packet memory on the Comet-TCP is not full. Then Comet-TCP encapsulate TCP packets into fixed-length packets with special header and sends them to remote Comet-TCP under QoS packet transmission rate control. When Comet-TCP receives packets whose transmission rate is more than the pre-determined value, packet transmission is paced by inserting gaps between packets. When the remote Comet-TCP receives packets, Comet-TCP recovers TCP packets and recovered packets are immediately sent to TCP/IP layer of the remote host processor. When the received TCP packet is the proper packet, the TCP layer on the remote host sends back ACK to the sender. When the Comet-TCP at the sender side receives the ACK, it removed packets stored in the buffer memory on Comet-TCP network. When the buffer memory on the local CometTCP becomes full, CometTCP does not return ACKs to the TCP driver and thus TCP driver stop sending packet until the buffer memory becomes available These functions are based on RFC 3135, “Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations” [1].

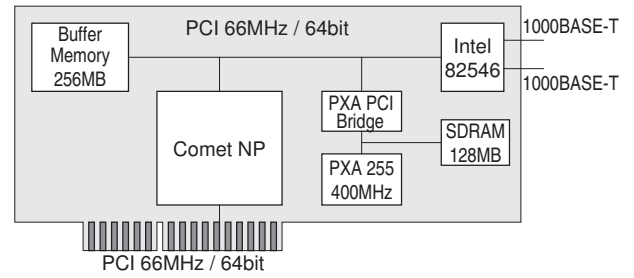


Figure 8: Block Diagram of Comet-TCP network interface card

Figure 8 shows block diagram of Comet-TCP network interface card. Comet-TCP is a network interface card with PCI 66MHz/64bit bus interface. Comet NP is a micro-programmed network processor that can transform packet streams and encrypt / decrypt packets into IPsec packet at wire speed of Gigabit Ethernet [10]. Buffer memory is used for storing transmitting packets and SDRAM is used for storing programs for strong-ARM (PXA255) control processor. Comet-TCP has two Gigabit Ethernet interfaces but a single interface is used for implementation of Comet-TCP.

4. TRANSMISSION RATE CONTROLLED TCP (TRC-TCP)

“Transmission Rate Controlled TCP (TRC-TCP)” is to relax the burstiness of TCP by approximating microscopic packet-level transmission rate to macroscopic flow-level transmission rate [16]. TRC-TCP is a combination of data-link layer approach and transport layer approach; *IPG control*, which is done by controlling network adapter’s transmission rate of packets, and *Clustered Packet Spacing*, which uses pseudo software interrupt function ‘tasklet’ in Linux kernel.

4.1 Inter Packet Gap (IPG) tuning

According to the IEEE’s Ethernet standard, an Ethernet adapter should insert a delay between successive transmissions of packets and this delay is called ‘Inter Packet Gap’ (IPG). A larger IPG will decrease the effective throughput and then the IPG can be used to control the aggressiveness how the sender emits data into network. Many Ethernet adapters can be configurable for IPG parameter by software. As for an Intel GbE adapter, the IPG value ranges from 8 to 1023 bytes in increments of 1 byte. We modify the Linux’s driver called ‘e1000’ to make it configurable for IPG using ‘ethtool’, and the default value is minimum 8 bytes.

Figure 9 shows the maximum, minimum and average throughput when we change the IPG from 8 to 1023 bytes. It’s true that IPG tuning is effective to make bursty behavior of GbE milder, i.e., busy Δt becomes $(IPG + MTU)/MTU$ times longer, when we assume one packet is about MTU long. But shortcoming of IPG tuning is that it affects all communication over LFT-tuned interface.

4.2 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” na-

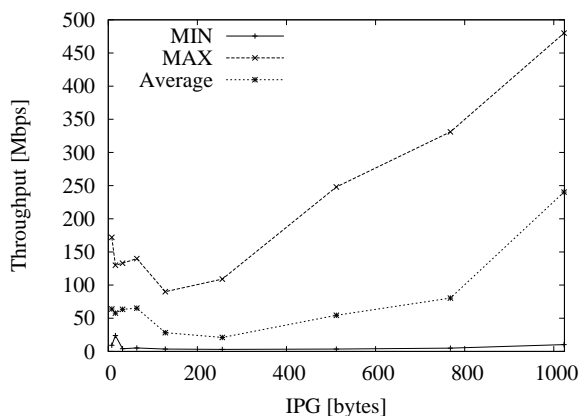


Figure 9: IPG and transfer rate with GbE RTT=198ms

ture but the network can't help pacing for high bandwidth-delay-product flows, it's important to distribute packets over one RTT until *cwnd* grows to appropriate size.

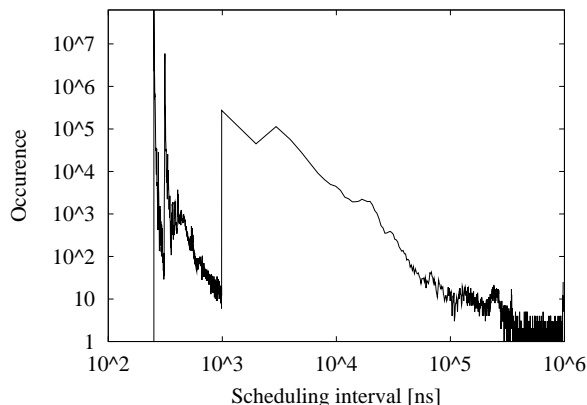


Figure 10: Histogram of our high resolution timer. During transferring a on-disk file on Linux 2.6

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 implement this method using pseudo software interrupt function 'tasklet' 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. To schedule packet transmission in target interval, we use tasklet to implement high resolution timer that triggers TCP's data transmission function. Figure 10 shows the resolution of our high resolution timer using Linux 2.6. The tasklet can be scheduled at least every $1 \mu s$.

Figure 12 shows how well TRC-TCP can approximate the packet-level rate to the flow-level rate during slow start, while normal TCP sends packets in burst at the beginning of every RTT (Figure 11).

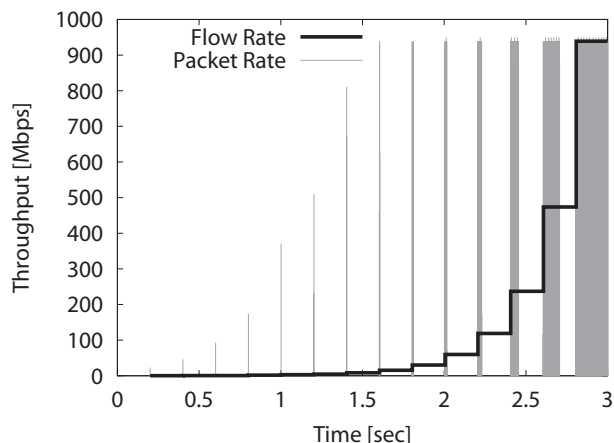


Figure 11: Flow- and packet-level rate of TCP during slow start. RTT is 200 ms

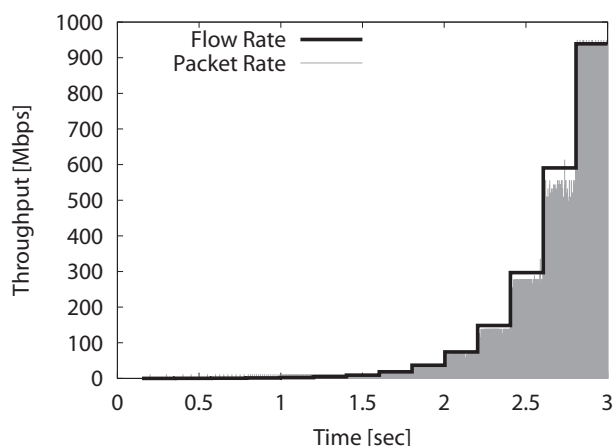


Figure 12: Flow- and packet-level rate of TRC-TCP during slow start. RTT is 200 ms

5. DULLING EDGES OF COOPERATIVE PARALLEL STREAMS (DECP)

"Dulling Edges of Cooperative Parallel Streams (DECP)" is to smooth the speed of parallel streams.

5.1 Basic Design

It is natural idea to utilize information of other streams to unify the recognition of network condition and make performance of streams balanced. But, we have to carefully design the way to exchange the information to avoid needless overhead, because, for example, a packet is processed every $12.5 \mu s$ for 1 Gbps Ethernet. In addition, we want to keep current TCP's AIMD framework including its parameters in this work; i.e., the influence of our modification is limited to the performance on parallel streams which user explicitly specifies. For this reason, we select polling to get information of all streams and add the interface on TCP to get and set information. We implement a simple external scheduler, whose policy can be changed very easily, even while in the middle of parallel data transfer.

As we have seen in section 1, data, which is sent but not ACKed, is called “inflight”, and its maximum size is called “window size”. To control throughput of each stream, Linux TCP kernel has variables, “`awnd`” Advertised Window size, and “`cwnd`” Congestion Window size. The former is set by the receiver and the latter is set by the sender. And, window size is set to the smaller value of either `awnd` or `cwnd`. To avoid any unexpected side-effect, we introduce a new variable `uwnd`, User Window size, which can be specified by user-land. `uwnd` is independent from `cwnd` or `awnd`, and window size is decided using `uwnd`, `cwnd`, and `awnd`.

We prepare a server-client system; a client program gets information “inflight” of its all streams, and a server program of the cluster collects information from their clients and notify summarized information to client program, in turn, then the client program set the information `uwnd` to its streams.

5.2 TCP kernel interface Implementation

To get information of window size of TCP connection, `inflight`, and to set information from user-land `uwnd`, we add entries to `set/getsockopt()` system calls. `uwnd` is currently a structure consists of the members `uwnd.max` and `uwnd.min`, so that user can specify range of preferable window size of streams. A variable `uwnd` is independent from `cwnd`, and is only used in testing function of packet sending; e.g., `tcp_snd_test()` of Linux. When `cwnd` is smaller than `uwnd.min` or `cwnd` is bigger than `uwnd.max` `uwnd.min` or `uwnd.max` is used instead of `cwnd`. We also need to name a group of TCP connections.

The added entries to `get/setsockopt()` is

TCP_ATTACHGROUP Set group name of TCP connection. (set only)

TCP_WININFO Argument is group name.

- `get` – return array of inflight information of each connection of specified group.
- `set` – set upper and lower limit of window size of each connection of specified group. If 0, no limit is set.

5.3 External Scheduler and window size control

We implement simple server-client system; one server for a cluster and one client for each node of the cluster. First, clients set group name to streams using `TCP_ATTACHGROUP`. Then, clients periodically get information of streams using `getsockopt(TCP_WININFO)` function, and report them to the server. Then the server computes preferable range of window size for streams, and notify this information to clients, in turn. Then, clients set the range of window size via `uwnd.max` and `uwnd.min` using `TCP_WININFO`. These programs can start and finish at anytime without any side effect.

Once `uwnd` is set, as we have described before, a packet test routine, `tcp_snd_test()`, checks whether window size is within the range which user specifies, and when it’s out of the range, the value of `uwnd` is used instead. When `uwnd.max` is same as `uwnd.min`, window size of all streams is set to same value.

Now, the question is how to find out preferable range of window size by collected information, i.e., inflight of parallel streams. We compare algorithms on real LFN by changing server program in user-land.

5.4 Preliminary test and Comparison of algorithms

Figure 13 and 14 shows the result when using 2.4 Gbps line without limit, i.e. using normal TCP. We use a pair of 8 IA servers and each IA server has 4 connections, so 32 connections are made in total. Figure 13 shows the total inflights over time of 32 streams and Figure 14 shows the maximum and minimum inflights.

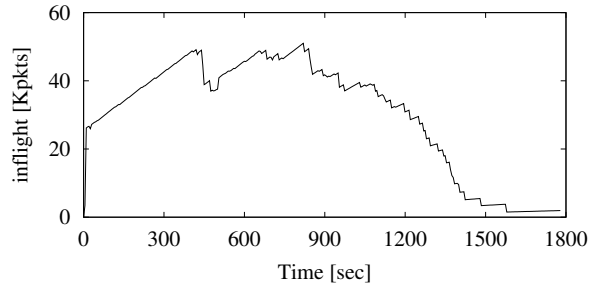


Figure 13: Total inflights over time of 32 normal streams from 8 nodes on OC-48

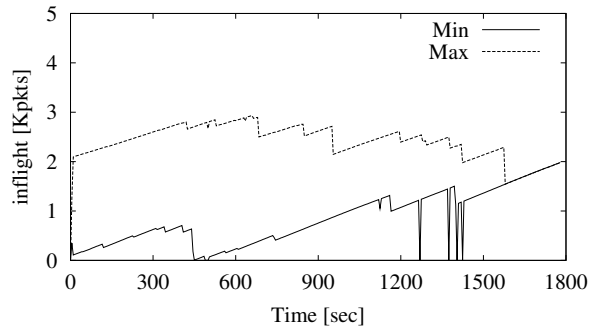


Figure 14: Maximum and minimum inflights over time of 32 normal streams from 8 nodes on OC-48

Figure 13 shows that the total throughput decreases after the one thirds of total time passes. This performance decrement is caused by that the fastest stream finishes its data transfer. After this, each time a connection finishes, throughput decreases. On the other hand, figure 14 shows that the maximum and minimum inflights of all connections differs about 4 to 5 times at the beginning of data transfer. But when the total throughput is decreasing, that is, after some connections has finished, maximum and minimum inflights get closer. After about 85% of total time has passed, only one connection remains. From this experiment, parallel streams of normal TCP fail to use bandwidth effectively for about 30% of the total execution time, and lower connections makes total communication time increase, and average throughput decreases.

Roughly, streams are categorized into two groups; fast connections and slow connections. Since slow connections cannot recover due to fast connections, the total throughput is not high. Once unbalance of window size occurs, it needs a lot of time to recover. Fast connections give a burden to networks, especially when the route is same, and it gives

bad effects to other connections. But the judgment whether it's too fast or not is difficult. In addition, if window size is too big, the penalty is big accordingly when packet loss occurs. The target of our algorithm is to suppress excessive connections and realize the normalized transfer rate.

We compare several naive algorithms such as set largest, larger, average value to all clients, or set upper or lower limit using average or weighted average of window sizes. Surprisingly, setting larger value or setting lower limit seems to make the performance worse. For example, if each client takes the maximum window size of all streams as its window size, the performance decreases very badly. As for our experiments for SC2003, we didn't set lower limit and only use upper limit.

5.5 Dulling Edges of Cooperative Parallel Streams (DECP)

To improve total performance of a system using parallel streams, we take a strategy that decelerates faster streams, in order to help slower streams to catch up quickly and reliably. Then, we compare several naive algorithms how to co-operate parallel streams, and decide the policy of algorithm. We adopt majority decision algorithm using scoreboard with the history, in order to avoid being too sensitive to the slowest stream or latest packet loss event. First, we quantize a value of inflight and divide into several blocks and record collected inflights using so-called "scoreboard" table. Clients send notification of the value of inflight periodically to the server.

- (1) Periodically, a server counts up the points of the scoreboard. First, all score is decreased with a fixed rate.
- (2) According to an inflight value sent from a client, a score is added in the convex form. It means that the exact block gets full points and the neighboring blocks also get partial points.
- (3) If the score of all connections are added, the block of highest score is taken as the most favorable block, and the corresponding window size is used as a goal.
- (4) A bit larger value than selected window size is notified as new upper limit to clients.

By using such kind of scoreboard algorithm, it's possible to increase the upper limit of window size gradually while waiting for most connections to grow up.

For this experiment, a window size is quantized by 30 packets and counted every 1 second. We use the decreasing ratio of score of 10% and 3 points for an exact block, 2 points for neighbors and 1 point for neighbors' neighbors. The value with extra 60 packets is notified to clients and they select the smallest from their congestion window, advertised window and the notified window size.

6. EXPERIMENTAL RESULTS

6.1 Network and Hardware Environment

For the SC2003 Bandwidth Challenge, we used four trans-Pacific OC-48 lines in addition to an trans-Pacific OC-192 line between Tokyo and Portland provided by IEEAF² used as loop-back circuit (Figure 15). Two of them are between Tokyo and Seattle provided by NTT Communications (NTT-C), which can be used exclusively. One is between Tokyo

²Internet Educational Equal Access Foundation. <http://www.ieeaf.org/>

and Los Angeles provided by APAN, which is connected to our servers via three non-aggregated GbEs. The last one is between Tokyo and New York provided by NII³, which is connected to our servers via one GbE. These lines are peered with Abilene. The aggregate bandwidth is 8.2 Gbps. Since data goes through the Tokyo-Portland loop-back on the way between Tokyo and Phoenix, the distance is 15,000 miles (24,000 km) and the RTT is 350 ms.

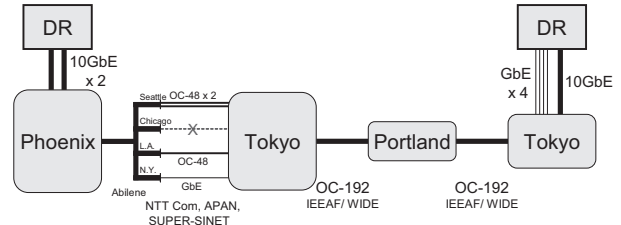


Figure 15: 15,000miles Network, 3 different routes merged to 8.2Gbps

As for Data Reservoir system, we use IBM x345 that consists of Dual Intel Xeon 2.40 GHz, 2 GB memory, Intel 82546EB GbE adapter, Redhat Linux 7.3, Kernel 2.4.18, and four 146 GB 10,000rpm Ultra320 HDDs.

6.2 SC2003 Bandwidth Challenge

6.2.1 Comet-TCP

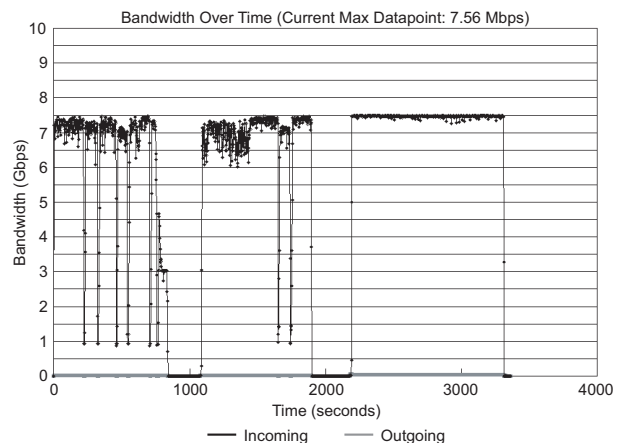


Figure 16: Throughput for Bandwidth Challenge (Comet-TCP)

Figure 16 shows performance result of 15,000 miles data transfer using a pair of 16 disk servers. Unstable performance behavior before 2000 seconds is caused by frequent temporary cut-down of the network between Tokyo, Japan and Portland, U.S. This kind of unstable network is difficult situation for TCP because temporary cut-down reset the size of congestion window and growth rate of the congestion window size is very slow on high-latency network. However, Comet-TCP regain almost maximum bandwidth right after temporary cut-down of the network because the size of

³National Institute of Informatics. <http://www.nii.ac.jp/>

congestion window can grow very rapidly because latency from TCP driver and local Comet-TCP is very small. After 2000 second, performance behavior became stable and 7.56 Gbps peak bandwidth was observed. This peak bandwidth is about 92% of available bandwidth (8.2Gbps) between two points.

6.2.2 TRC-TCP

For TRC-TCP, we use Data reservoir system consists of 16 disk servers and each node makes 4 streams, and this means 64 parallel streams within one system. These nodes are divided into 4 nodes for each NTT-C's OC-48, 6 nodes for APAN's OC-48 and 2 nodes for NII's GbE. 6 nodes for APAN are further divided into 2 nodes for each GbE.

Figure 17 shows the throughput over time we measured by switches. Figure 18 shows the total size of inflights of each line.

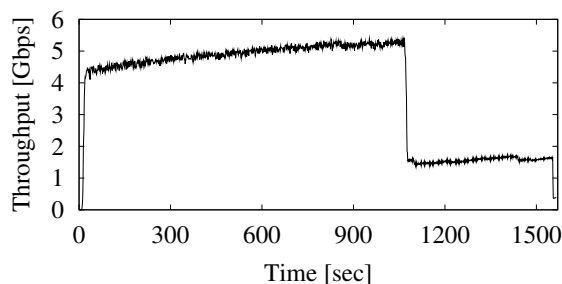


Figure 17: Throughput for Bandwidth Challenge (TRC-TCP)

The official record is found at <http://scinet.supercomp.org/2003/bwc/results/>

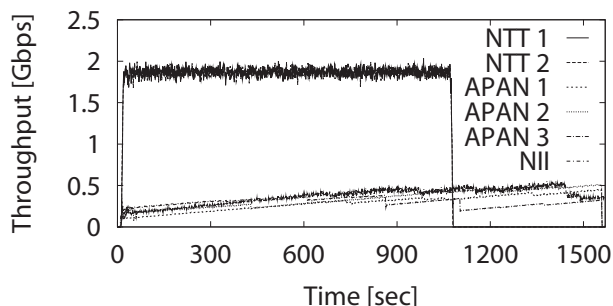


Figure 18: Total inflights of TRC-TCP streams for Bandwidth Challenge on each line. NTT 1/2: 16 streams from 4 nodes on OC-48, APAN: 24 streams from 6 nodes on OC-48, NII: 8 streams from 2 nodes on GbE

6.2.3 DECP

DR system consists of 32 disk servers and each node makes 4 streams, and this means 128 parallel streams within one system. These nodes are divided into 10 nodes for each NTT-C's OC-48, 9 nodes for APAN's OC-48 and 3 nodes for NII's GbE. 9 nodes for APAN are further divided into 3 nodes for each GbE.

Figure 19 shows the throughput over time we measured by switches. The official maximum instantaneous throughput is 7.01 Gbps. Figure 20 shows the total size of inflights of each line.

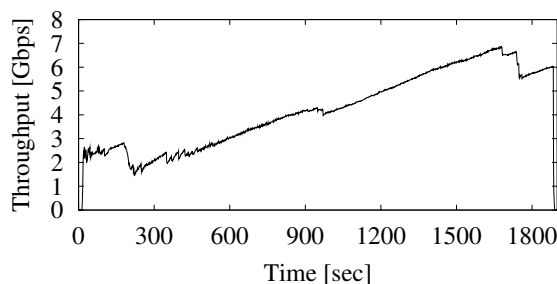


Figure 19: Throughput for Bandwidth Challenge

The official record is found at <http://scinet.supercomp.org/2003/bwc/results/>

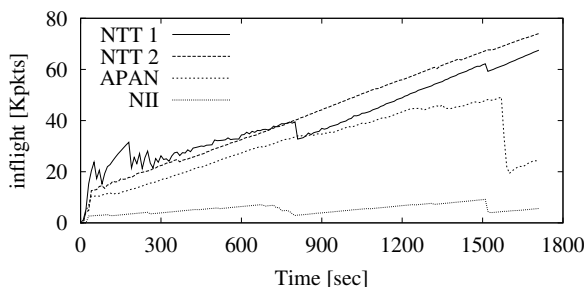


Figure 20: Total inflights of DECP streams for Bandwidth Challenge on each line. NTT 1/2: 44 streams from 11 nodes on OC-48, APAN: 36 streams from 9 nodes on OC-48, NII: 12 streams from 3 nodes on GbE

6.3 Discussion

To our surprise, both TRC-TCP and DECP can almost keep growing its transfer rate gradually until some of the streams finish transfer of their striped data. They succeed to alleviate rate dispersion.

Actually, it's strange that DECP doesn't fail to grow, since flow-level rate and packet-level rate is different and self-made congestion likely occurs. One of the biggest reasons might be that bottleneck lines can be used almost exclusively and packet loss is rare. But still, it's obvious that As for TRC-TCP, it grows for more than 15 minutes, and maximum usage of bandwidth is about 80%. As for DECP, it grows for more than 30 minutes, and maximum usage of bandwidth is more than 85%. The reason why growing speed of DECP is faster than TRC-TCP is because the number of streams is twice.

7. RELATED WORK

The idea of limiting the packet transmission rate from a TCP source host is shown, for instance, in [11]. Efficient use of high-bandwidth network with long latency has been one

of central issues for realizing global data sharing and useful grid computing. Methods to realize efficient but network friendly use of LFN can be classified into three categories; i.e. (1) improvement of single TCP stream, especially TCP congestion control mechanism, (2) use of parallel and multi TCP streams, and (3) instrumentation methods to analyze actual behavior of LFN.

Progress in control of congestion window size and retransmission in TCP have been improved during past ten years such as TCP NewReno[8] and TCP Vegas [2] and extension of these protocols. Recently, with the rapid growth of latest optical networking technology, several works try to meet LFN by changing window size more effectively than standard TCP, such as Scalable TCP (STCP) [12], High Speed TCP (HSTCP) [4] and FAST TCP [9]. One big difference of our work and these works is that they assume that congestion can be guessed properly by current scheme; i.e., *ACK* or *RTT*, but we don't. We believe that the difference between flow-level rate and packet-level rate or parallel streams cause self-made congestion which can be sometimes avoidable, and we try to reduce needless packet losses by pacing or slow down. And, our technique can be used with other window size control mechanisms just same as current TCP. Pacing has been studied well in the field of multimedia communication, where response time is important [13].

Hacker studies parallel streams using theoretical model [5]. P.Sockets [17] is an implementation of parallel TCP. It offers the user level library written by C++ and users can avail them without system parameter modification. But, since striping and buffering is done by its library, it cannot treat well direct communication with smaller number of memory copies or striped data in several hosts. There exist applications such as GridFTP [21], bbftp [19] and pftp [20]. Hacker also proposes to manage parallel streams by suppressing growing speed of window size to keep fairness [6]. Ensemble-TCP is to make TCP Reno like parallel streams [3]. The biggest difference of our DECP and others is that we have an external scheduler to use information of window size of other machines; i.e., information inside kernel of other hosts.

8. CONCLUDING REMARKS

In this paper, we show that LFN is not a straight pipe of high-bandwidth network but a complicated network that is lossy on bursty traffic of TCP workloads. We show Comet-TCP, TRC-TCP and DECP are effective to improve efficiency and stability of TCP on LFN. Our hardware approach use 90% of bottleneck bandwidth and software approach attains max 85% of bottleneck bandwidth on real network of 15,000 miles, which is longer than going half around the globe. At a glance, what we do seems to be negative; i.e., both Comet-TCP and TRC-TCP puts additional interval between packets and DECP executes slow down of the faster stream among parallel streams, which after all reduces needless self-made congestion.

Current Comet-TCP is not fit to be used in the ordinary Internet environment because they cannot share the bandwidth in friendly way. But, still, Comet-TCP can work well in the situation such that when a quantity of bandwidth is allocated in advance. One of good points of TRC-TCP and DECP is, both are independent from window size control parameters and can be applied with most existing methods. We evaluate both methods one by one, due to the time limitation of experiments on LFN; actually we can use 8.2

Gbps bandwidth only for 15 hours (5 hours per day for 3 days) because of the fiber trouble under the sea, which occurred just before the experiments. As for TRC-TCP, since congestion window size changes dynamically, it is preferable that packet interval can be changed dynamically along congestion window size. The packet interval of GbE is about 12.5 μ s and that of 10GbE is 1.25 μ s for 1500-byte packets, thus, to solve this problem, it must be reasonable to use a kind of TCP offload engine; i.e., partial information of transport layer is informed to intelligent network card of link layer, which has buffer for each stream and is informed congestion window size and *RTT* by its host. As for DECP, due to the limitation of current implementation, which can't change congestion window size gradually, DECP converges slowly but steadily, but adjusting the parameter of additive increase makes convergence of streams' speed faster. In addition, DECP brings scalability to the system of parallel streams by equally sharing the bandwidth with negligible overhead.

For inter-layer cooperation, designing how to notify information over layers with very small overhead is important, especially when we use hardware. To develop network card which automatically control its packet-level rate using information of parallel streams is one of our future works.

Acknowledgments

Special thanks to Prof. Akira Kato of University of Tokyo for useful discussions and arranging long distance experiments. We thank Mr. Furukawa, Mr. Yanagisawa, Mr. Nakano, Mr. Torii, and Mr. Mizuguchi of Fujitsu Computer Technologies, and Dr. Matsuo, Dr. Masuoka of Fujitsu Laboratory America. This research 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 (STAR) Japan, CREST project of Japan Science and Technology Corporation, and by 21st century COE project of Japan Society for the Promotion of Science. The data transfer experiment is supported by NTT Communications Inc., IEEAF, APAN, Wide Project, NII, Tyco Telecommunications, Juniper Networks, Inc. Cisco Systems Co. Ltd., NetOne systems Co., Ltd. and Digital Technology Co., Ltd.

9. REFERENCES

- [1] J. Border, M. Kojo, J. Griner, G. Montenegro, Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations", RFC 3135, (2001)
- [2] 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.
- [3] L. Eggert, J. Heidemann, and J. Touch, "Effects of Ensemble-TCP", ACM Computer Communication Review, 30 (1), pp.15-29, Jan. 2000.
- [4] S. Floyd, "HighSpeed TCP for Large Congestion Windows", RFC 3649, Dec. 2003.
- [5] 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.

- [6] T. Hacker, B. Noble and B. Athey, "Improving Throughput and Maintaining Fairness using Parallel TCP", IEEE Infocom 2004, Mar. 2004.
- [7] 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>
- [8] J. C. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", Proc. ACM SIGCOMM '96, pp.270-280, Aug. 1996.
- [9] C. Jin, D. Wei, and S. Low "FAST TCP: Motivation, Architecture, Algorithms, Performance", IEEE Infocom 2004, Mar. 2004.
- [10] A. Jinzaki, "Stream Processor", Proc. JSP2000, pp. 205-212, (in Japanese), 2000.
- [11] Shrikrishana Karandikar, S. Kalyanaraman, P. Bagal, and B. Packer "TCP Rate Control," ACM Computer Communications Review, Jan. 2000.
- [12] T. Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks", PFLDnet 2003.
- [13] 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 .
- [14] 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.
- [15] S. Nakano, and et al. "DR Giga Analyzer", Symp. on Global Dependable Information Infrastructure, Feb. 2004. (in Japanese)
- [16] M. Nakamura, J. Senbon, Y. Sugawara, T. Itoh, M. Inaba, K. Hiraki, "End-node transmission rate control kind to intermediate routers - towards 10Gbps era", PFLDnet2004, Feb. 2004.
- [17] 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.
- [18] Y. Sugawara, M. Inaba, and K. Hiraki, "Over 10Gbps String Matching Mechanism for Multi-Stream Packet Scanning Systems" FPL2004, To appear.
- [19] : BBftp.
<http://doc.in2p3.fr/bbftp/>.
- [20] : pftp.
<http://www.indiana.edu/rats/research/hsi/>.
- [21] : GridFTP.
<http://www.globus.org/datagrid/>.