Performance of high-speed transport protocols coexisting on a long distance 10-Gbps testbed network

Kazumi Kumazoe
NICT, Kyushu Research Center
3-8-1, Asano, Kitakyushu
Fukuoka Pref., Japan
802-0001
kuma@kyushu.jgn2.jp

Masato Tsuru
Kyushu Institute of Technology
280-4, Kawazu, Iizuka
Fukuoka Pref., Japan
820-8502
tsuru@ndrc.kyutech.ac.jp

Yuji Oie
Kyushu Institute of Technology
280-4, Kawazu, Iizuka
Fukuoka Pref., Japan
820-8502
oie@cse.kyutech.ac.jp

ABSTRACT

Our group has been conducting a series of experiments on various recently available high-speed transport protocols over long-distance, 10-gigabit-class, international testbed networks. The study reported here focuses on the scenario of multiple coexisting flows along an end-to-end 10-Gbps path, where TCP-based transport protocols are used. The results show that two coexisting high-speed flows cannot share network resources efficiently. For example, for some protocols, competition between two flows prevents the sum of their throughputs from fully utilizing available bandwidth, especially when flows have the same RTT, and causes considerable imbalance in the time-averaged throughput between two flows, especially when there are different RTTs.

Keywords

high-speed transport protocol, Grid computing, high-speed Internet

1. INTRODUCTION

A variety of high-speed transport protocols have been proposed in response to emerging requirements for extremely high throughput data transfer across fast long-distance networks in distributed data processing and data sharing environments such as Grid computing. The bandwidth of global networks like the Internet, however, has been increasing in both access and core networks. In Japan, for example, reasonably priced 1-Gbps broadband access (FTTH) services have recently become available, and 10-Gbps services might be available soon. In addition, recent off-the-shelf high-end computers combined with 10-Gbps NIC can in practice archive more than 5 Gbps throughput. Thus, Grid users will probably want high-speed transport protocols on an end-to-end long distance path to transfer a larger amount of data.

Various high-speed transport protocols have already been proposed, and research groups have been conducting a variety of experiments focusing on these protocols for high-throughput data transfer in fast long-distance networks using simulators, emulators, and testbed networks. Among them, high-speed testbed networks have relatively realistic characteristics (production-level router mechanisms and configurations, physical distance, etc.) compared with emulator or simulator networks, while they allow us to investigate the internal status and/or to control competing traffic to some extent. For example, the Hamilton Institute group made a common benchmarking environment using dummynet (network emulator), where the different protocols were implemented with common network stacks and evaluated in terms of a common performance measure[10][3]. The BIC Lab team tried to realistically evaluate performance by creating an emulator-based network environment in which high-speed transport protocol flows are accompanied by well-designed background traffic[2][12]. For benchmarking in 10-gigabit-class environments, the Caltech group is constructing the Wan-in-Lab (WiL), which is a 2400-km long-haul fiber optic testbed, in a single laboratory as an open benchmark testing platform for the research community to evaluate different protocols in a more realistic environment[14].

We also started investigating several promising high-speed transport protocols through experiments using the Japan Gigabit Network II (JGNII), which is an open 10-Gbps-class Ethernet-based network between the USA and Japan. In our previous studies [7][8], we reported a series of experiments with 1-Gbps end equipment (sender and receiver hosts) and a 1-Gbps bottleneck link. We did this by examining what happened in several typical scenarios, such as when there are a change of network routes, coexisting short-lived TCP flows (e.g., web browsing flows), and coexisting constant bit-rate UDP flows (e.g., video stream flows). We also developed a TCP-related monitoring tool for internally monitoring kernels in these experiments[9].

Despite these evaluations of high-speed transport protocols, there has been little detailed reporting on the problems faced by high-speed flows competing on a 10-Gbps path even though such long distance global networks must be shared by various users and applications. Thus, in this study we experimentally investigated the actual behaviors of a few high-speed flows competing on an end-to-end 10-Gbps-class network. We focused in particular on how to fully exploit the 10-Gbps bandwidth and how to stably balance those flows

GridNets 2007, October 17-19, 2007, Lyon, France

DOI 10.4108/gridnets.2007.2248

in their time-average throughputs, both of which issues are of practical importance for Grid computing.

The remainder of this paper is organized as follows. Section 2 describes the experimental environments. Section 3 presents the experimental results, and Section 4 makes some closing remarks.

2. EXPERIMENTAL SETUP

The network configurations used in our experiments are shown in Fig. 1. In Case (a), Flows 1 and 2 were established between Kitakyushu, Japan and Chicago, USA. Each flow had a measured round trip time (RTT) of 180 ms. In Case (b), Flow 1 was established between Kitakyushu and Chicago, while Flow 2 was established between Kitakyushu and Tokyo (measured RTT = 18 ms). We considered several coexisting flow scenarios: flows from a single sender to multiple clients (i.e., sender machine bottleneck), flows from different senders to multiple clients (i.e., network bottleneck), and single or multiple flows between a server/client pair. We also examined the impact of socket buffer sizes and of a TCP offload engine. Sender (server) and receiver (client) equipment specifications are listed in Table 1.

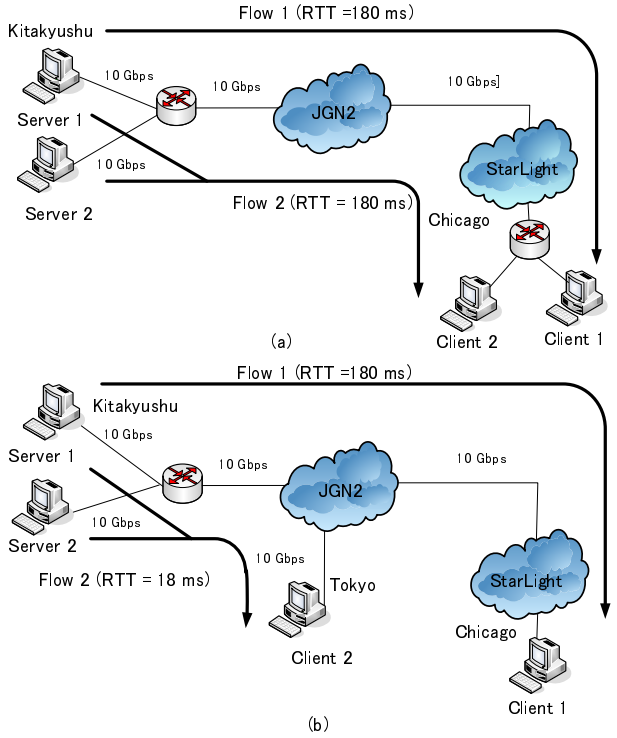

Figure 1: Network configurations. Top: (a) data transfer between Kitakyushu and Chicago; bottom: (b) data transfer between Kitakyushu-Tokyo and Kitakyushu-Chicago

We used Linux (SUSE Linux kernel 2.6.16) as the sender-side OS in the experiments. Kernel 2.6.16 supports pluggable congestion control algorithms by setting the sysctl variables. In this study, we focused on Standard TCP (RENO), HSTCP, Scalable TCP, HTCP, CUBIC, and Compound TCP. Patch code for Compound TCP is provided in [11]. Other targeted protocols were originally implemented in kernel 2.6.16.

We tuned various parameters (e.g., Linux txqueuelen and system memory) based on technical information [1][13][15] [6][4], as shown in Fig. 2.

The performance of high-speed transport protocol is significantly affected by the buffer size of the bottleneck link (router), which has several internal output buffers with a complex architecture. Therefore, we present experimental results obtained when the buffer size at the ingress edge router (Hitachi GS 4000) was configured to be as large as possible. We also monitored the packet loss counter at the edge router during data transfer. Note that the JGNII paths used in our experiment share network resources with other constant or temporary traffic on the JGNII. To prevent such unexpected and uncontrollable concurrent background traffic from influencing our results, we conducted several trials (at least three times) for each experiment with the same configuration, and then discarded irregular cases. Unless otherwise noted, the results shown are for cases with typical or relatively good throughput performance, which is expected when the influence of background traffic is negligibly weak.

```
sysctl -w net.ipv4.route.flush=1

### for large window
sysctl -w net.core.rmem_max=268435456
sysctl -w net.core.wmem_max=268435456
sysctl -w net.core.rmem_default=65536
sysctl -w net.core.wmem_default=65536
sysctl -w net.ipv4.tcp_rmem="4096 268435456 268435456"
sysctl -w net.ipv4.tcp_wmem="4096 268435456 268435456"
sysctl -w net.ipv4.tcp_mem="268435456 268435456 268435456"
sysctl -w net.ipv4.tcp_no_metrics_save=1

sysctl -w net.core.netdev_max_backlog=300000

sysctl -w net.ipv4.tcp_tw_recycle=0
sysctl -w net.ipv4.tcp_tw_reuse=0
sysctl -w net.ipv4.tcp_sack=1
sysctl -w net.ipv4.tcp_timestamps=1

ifconfig eth1 mtu 9000
ifconfig eth1 txqueuelen 10000

### for neterion+opteron
setpci -d 17d5:5832 62=1d

sysctl -w net.ipv4.tcp_congestion_control=(reno, highspeed, bic,
cubic, htcp, compound)
```

Figure 2: Parameter tuning for TCP flows.

3. EXPERIMENTAL RESULTS

We show the throughput characteristics of a single flow using each high-speed transport protocol in subsection 3.1 and those of multiple coexisting flows in subsections 3.2–3.5. TCP throughputs are illustrated as a time series of one-second averaged goodputs measured by the receiver-side iperf[5] command. We also examine the impact of changing the socket buffer sizes (at both the sender and the receiver), which can be set by the iperf command "-w". Note that, in this paper, the socket buffer size is indicated by using "sock buf". Moreover, for Figs. 4, 7, and 9, we compare coexisting flows in the TCP throughput averaged over periods in which there are multiple coexisting flows. The throughput values are also averaged over several trials.

Table 1: Equipment specifications

	End host in Chicago	End host in Tokyo	End host in Kitakyushu
OS	Debian Linux	Debian Linux	SUSE Linux
CPU	Opteron 2.4 GHz	Opteron 2.4GHz	Opteron 2.8 GHz
Memory	1 GByte	1 GBytes	4 GBytes
PCI bus	64 bits		
NIC	Neterion Xframe II (2.0.18.8693)		

3.1 Throughput characteristics of a single flow

In our laboratory, we have two kinds of server with PCI-133 buses: HP-xw8200 and HP-xw9300. Each has one PCI-X 133 slot and two PCI-X 100 slots. We also have two kinds of 10-Gbps network interface card: Intel PRO/10GbE and Neterion Xframe II. Table 2 summarizes the maximum observed throughput of a single TCP flow (Flow 1) in Fig. 1.

Table 2: Maximum observed throughput of a single flow.

End host	PCI-X	NIC	TSO	Throughput [Gbps]
xw-8200	133	Xframe II	off	5.2
		PRO/10GbE	off	4.8
xw-9300	133	Xframe II	on	6.5
		PRO/10GbE	on	6.1
	100	Xframe II	on	5.8
			off	5.4
		PRO/10GbE	on	5.0

Using xw-8200 as a sender server at Kitakyushu, we observed that the maximum throughput of a single flow was 5.2 Gbps when the network interface card (NIC) was installed in the PCI-X 133 slot. This limitation must be due to the type of memory implemented in the xw-8200. We observed the highest throughput of a single flow when Xframe II was installed in the PCI-X 133 slot on xw-9300 machine and TSO (TCP segmentation offload) was on. In that case, the one-second averaged throughput measured by iperf reached 6.5 Gbps. We initially planned to use the PCI-X 133 bus on xw-9300. Unfortunately, the NIC (both PRO/10GbE and Xframe II) behaved erratically because the area around the PCI-bus slots became very hot when the NIC was installed in PCI-X 133.

One reason for this trouble is that the PCI-X 133 slot, which is the fastest bus of xw-9300, is located at the edge of its chassis, so it is difficult to dissipate the heat around the NIC. Therefore, we present the throughput characteristics of flows observed when the Xframe II card was installed in the PCI-X 100 slot on xw-9300. In this configuration, the maximum throughput of a single constant bit rate flow measured by iperf (with the UDP option) was 5.5 Gbps. Figure 3(a) shows the throughput changes of a single flow (Flow 1 shown in Fig. 1(a)) of Compound TCP when it ran alone for 60 seconds. We observed similar tendencies in throughput characteristics of all the other TCP-based transport protocol flows (including Reno flows). Figure 3 (a) shows the change of CPU utilization measured using vmstat. Figure 3 (b) shows the throughput characteristics of a single CUBIC flow for various socket buffer sizes.

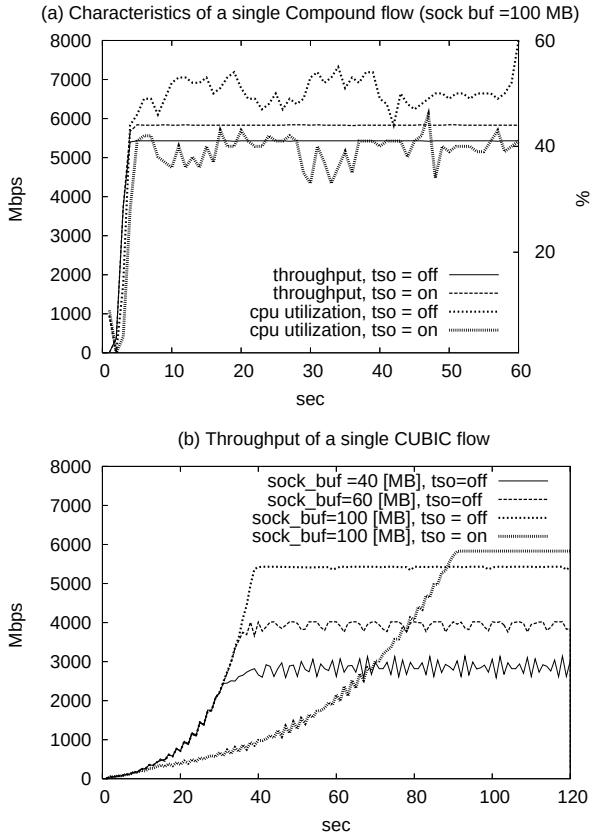

Figure 3: Characteristics of a single flow.

The following points can be observed from these figures. First, whenever the network is not congested, a single TCP flow can stably achieve maximum throughput (more than 5 Gbps), which might be limited by the hardware specifications. Second, an insufficient socket buffer size also limits its maximum throughput, as shown Figure 3 (b). Finally, higher throughput was achieved when TSO was on, as shown in Table2. In addition, the CPU utilization was relieved to some extent by setting TSO to on, as shown in Fig. 3(a). All the other protocols except for CUBIC behaved in a similar manner. The throughput of a CUBIC flow, however, increased slowly when TSO was on, in contrast with when it was off, as can be seen in Fig. 3(b).

Note that the use of TSO increases the time-averaged TCP throughput and decreases the CPU load to some extent, while also slowing down the increase in TCP throughput in CUBIC. To compare different protocol flows, hereinafter we show results obtained when TSO is set to off, unless otherwise noted.

3.2 Data transfer from a single server

We examined the throughput characteristics of multiple coexisting flows using the same transport protocol in scenarios where multiple flows ran from a single sender (Server 1), i.e., a potential bottleneck (= 5.4 Gbps).

Figure 4 summarizes the time-averaged throughput of each individual flow from a single server using various high-speed transport protocols. Case (1) shows the averaged throughput of a single flow running alone, Case (2) indicates the throughput of four coexisting flows that are established using iperf "-P 2" command between both Server 1 – Client 1 and Server 1 – Client 2 pairs simultaneously in Fig. 1(a). Case (3) indicates throughputs for four flows in Fig. 1(b) where Flows 1 and 2 were established between Server 1 and Client 1 and Flows 3 and 4 were simultaneously established between Server 1 and Client 2 using iperf "-P 2" command.

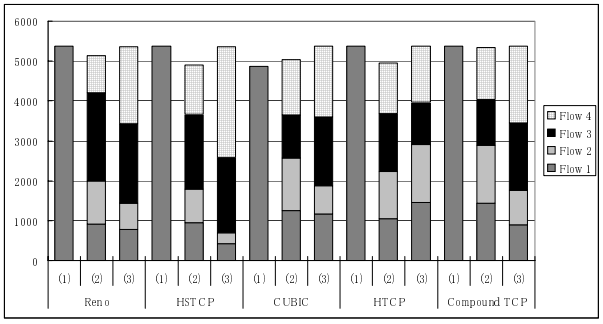

Figure 4: Data transfer from a single server.

It can be observed in Case (2) that when multiple flows have the same RTT they shared resources fairly when they started simultaneously, except when Reno and HSTCP protocols were used. In Case (3), where Flows 1 and 2 have RTTs ten-times longer than those of Flows 3 and 4, the advantage of shorter RTT flows (Flows 3 and 4) is obvious in Reno, HSTCP, CUBIC, and Compound TCP cases. That is, resources were shared unfairly between the coexisting flows. In the HTCP protocol, in contrast, coexisting flows shared resources fairly. In all cases, it can be observed that the maximum throughput limited by the server can be fully shared by multiple coexisting flows when one server sends its data to multiple clients simultaneously.

Next, we show the time series throughput characteristics in scenarios where two flows with different start times coexist. In the first scenario, the receiver hosts of two coexisting flows were located near each other (i.e. had the same RTT), as shown in Fig. 1(a). Flow 1, from Server 1 to Client 1, started transferring data first. Then, after 50 seconds, Flow 2, from Server 1 to Client 2, started transferring data, and continued to do so for 300 seconds. In all protocol flows, we observed that Flow 1's throughput was affected when Flow 2 started to transfer data. Flow 1's throughput level recovered to its ssthresh level and increased its cwnd in congestion avoidance mode. However, we observed a difference in the throughput behavior of Flow 2 with different kinds of high-speed transport protocols. As shown in Fig. 5(a), in every trial, when Flow 2 uses HTCP, its throughput increases relatively rapidly and throughput is shared fairly between the coexisting flows. With all the other protocols, Flow 2 increased its throughput relatively slowly and the sharing was unfair, as shown in Fig. 5(b).

Next, we investigated heterogeneous RTTs, where data was transferred from a single server to clients located in different places, as shown in Fig. 1(b). We considered two scenarios: in Scenario 1, Flow 1 started transferring data first. Fifty seconds later, Flow 2 started transferring data and continued to do so for 300 seconds, while in Scenario 2, Flow 1 started 50 seconds after Flow 2. Figure 6(a) shows the throughput characteristics for HTCP and Fig. 6(b) shows them for HSTCP. In both scenarios, the throughput characteristics of the flow that started first were affected when the new flow started and recovered to ssthresh, as in the case in Fig. 5. Flow 2, which started later in the case shown in Fig. 6, increased its throughput more quickly than Flow 2 in Fig. 5. This is because the RTT of Flow 2 in Fig. 6 is one tenth as long as that of Flow 1. In the HTCP case shown in Fig. 6(a), Flow 1, which had a ten-times longer RTT, achieved higher throughput than Flow 2, while for the HSTCP shown in Fig. 6(b), Flows 1 and 2 shared the bandwidth evenly. These tendencies stayed the same regardless of the socket buffer size and TSO setting.

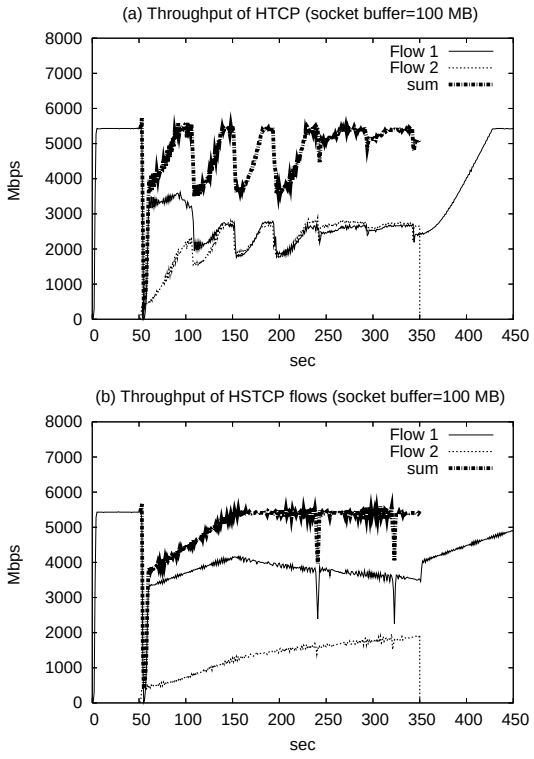

Figure 5: Data transfer from a single server.

3.3 Multiple coexisting flows with identical RTT

We examined the throughput characteristics of multiple coexisting flows using the same transport protocol in scenarios where flows ran from Server 1 to Client 1 and from Server 2 to Client 2, as shown in Fig. 1(a), where multiple flows have the same RTT and the bottleneck might be in the ingress router close to the senders. Hereafter, we observed the counter of packet loss of the router, which accommodates Servers 1 and 2 at Kitakyushu during observation of the throughput characteristics. We found that sometimes

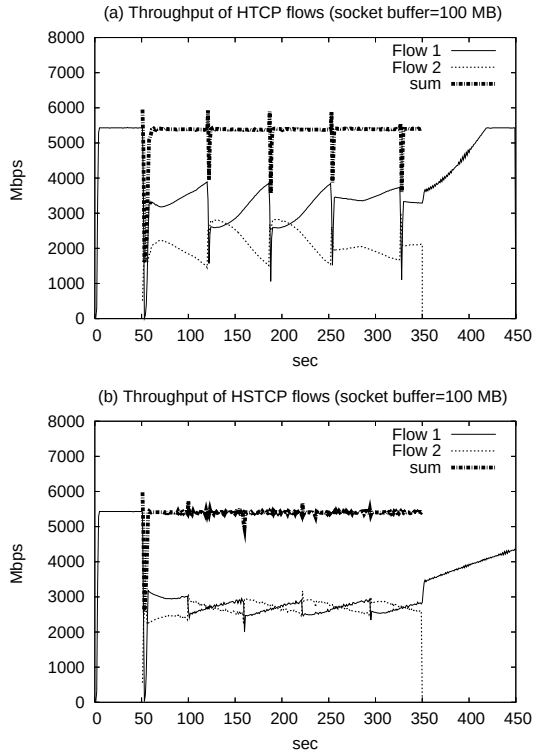

Figure 6: Data Transfer from a single server (different RTTs).

but not always the packet loss counter increases (for example, the observed packet loss rate at the switch was between $3e-6$ to $1e-5$) when the throughput decreases. Thus the bottleneck points might be not only the router at Kitakyushu but also located another place along the path. We measured the total throughput of two concurrent UDP (constant bit rate) flows, Flows 1 and 2, at the receivers, and found it was about 9 Gbps without packet losses. That is, the bandwidth of the link can be guaranteed to be at least 9 Gbps. In this subsection, we discuss our evaluations of two cases of socket buffer size: in Case (L), the socket buffer size was set to 100 MB so that the maximum throughput of a single flow could reach 5.4 Gbps, and in Case (S), it was set to 40 MB so that the throughput of a single flow was limited to 3 Gbps.

Figure 7 shows the total throughput of multiple flows established in Fig. 1(a) in Case (L). Multiple flows were simultaneously established between the server-client pairs (Server 1 – Client 1 and Server 2 – Client 2), using iperf "-P" command. Two, four, and six flows were established simultaneously. In case of coexisting multiple flows, the more the number of coexisting flows increased, the higher the total throughput to some extent. Interestingly, however, in all protocols the total throughput of six flows was smaller than that of a single flow, which imply that coexinting flows are adversely affected by each other. In addition, the more the number of coexisting flows increased, the smaller the throughput of each flow while the more the throughput fairness (balance) among flows improved, especially when HSTCP, CUBIC, and HTCP protocols were used.

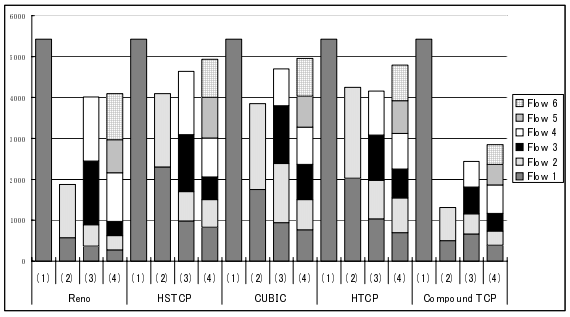

Figure 7: Time averaged throughputs in data transfer from two servers.

The time series throughput characteristics of Flows 1 and 2 in Case (L) with two coexisting flows and TSO off are shown in Fig. 8. In this scenario, Flow 1 started transferring data first, and Flow 2 started transferring 50 seconds later and continued for 300 seconds. We also examined these flows with TSO set to on. As can be seen in Fig. 8, when Flow 2 started at 50 s, the throughput characteristics of Flow 1 decreased in all protocols. CUBIC flow can recover to the original level quickly, while flows in other protocols recover to ssthresh.

We observed similar tendencies in the throughput characteristics of flows using Reno and HSTCP. Figure 8(a) presents flows for HSTCP. Soon after Flow 1 started, its throughput jumped to 5.4 Gbps. Then, after Flow 2 started, the throughput of Flow 1 fell to 3.8 Gbps and then gradually increased, and the throughput of Flow 2 increased very slowly. For example, the throughput of Flow 2 at 200 s was 400 Mbps. After Flow 2 stopped, the throughput of Flow 1 gradually increased. In Fig. 8(b) for CUBIC protocol, it took about 30 s for the throughput of Flow 1 to reach 5.4 Gbps, similarly in Fig. 3(b), while throughput of the Flow 2 took 130 s to reach 1 Gbps and 300 s to reach 1.85 Gbps. Then, after the total throughput of the two flows reached about 6 Gbps, the throughputs of both flows decreased simultaneously. After that, the throughput of Flow 1 was able to recover quickly to its original level, while that of Flow 2 recovered only slowly. The throughputs of the two coexisting HTCP flows increased and decreased simultaneously, as shown in Fig. 8(c). The total throughput of the two flows repeatedly increased to some extent and then decreased. After Flow 2 stopped transferring data at 350 s, the throughput of Flow 1 recovered quickly. Fig. 8(d) shows the fluctuation of the coexisting Compound TCP flows of throughputs in a short period. After Flow 2 stopped, the behavior of Flow 1 changed.

Next, we observed the throughput characteristics in Case (S) where the achievable one-second averaged throughput of a single flow was limited to 3 Gbps, so two flows would likely be able to coexist successfully without competing with each other. Based on experimental results not shown here, however, considerable interference was observed between two flows, which is similar to Case (L) in Fig. 8, although the maximum achievable throughput for each individual flow differs from that in Case (L). We also repeatedly examined the same scenarios when TSO was on with a variety of socket buffer sizes and, as in Cases (L) and (S), found that interference between those flows resulted in oscillation

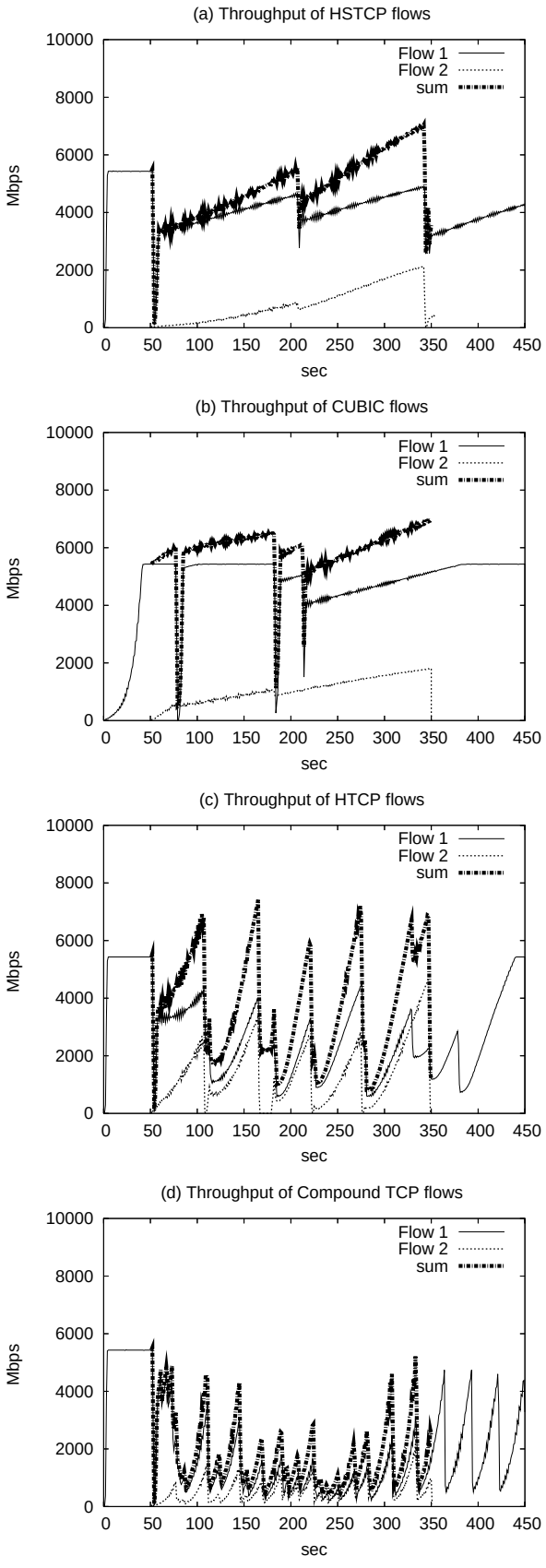

Figure 8: Data transfer from two servers in Fig.1 (a)

and inefficiency of throughput.

Consequently, two flows with identical RTTs coexisting in the 10-Gbps end-to-end path interfered with each other, and their throughput tended to increase and decrease simultaneously regardless of the socket buffer sizes and TSO setting. In contrast, in the 1-Gbps end-to-end path reported in our previous studies, this kind of interference could to some extent be avoided by limiting the socket buffer size. This might be because a 10-Gbps end-to-end path allows an extremely bursty transfer within a part of an RTT, especially in the slow-start phase. This cannot be mitigated by resetting the socket buffer size.

3.4 Two coexisting flows with different RTTs

We examined the scenario with two kinds of coexisting flows competing at the ingress router on the path, i.e., flows with RTTs of 180 ms and 18 ms, as shown in Fig. 1(b). We observed that the maximum total throughput per second of two flows was about 6-9 Gbps depending on the transport protocol.

Figure 9 summarizes the averaged throughput of coexisting flows in Fig. 1(b). Case (1) shows the averaged throughput of a single flow running alone (Flow 1), and Case (2) shows that for two coexisting flows, where Flow 1 was established between Server 1 and Client 1 and Flow 2 was established between Server 2 and Client 2. In Case (3), four flows were established simultaneously. Flows 1 and 2 are established between Server 1 and Client 1, while Flows 3 and 4 are established between Server 2 and Client 2.

By comparing Cases (2) and (3) in Fig. 9 to Cases (2) and (3) in Fig. 7, it can be observed that the different RTTs of the coexisting flows help to mitigate the throughput degradation by interference, while flows that have shorter RTTs have a great advantage in throughput competition. As observed in Fig. 9, the difference between averaged throughput characteristics in flows that have different RTTs is obvious.Throughput of flows that have longer RTTs using Compound TCP was especially limited in our configuration.

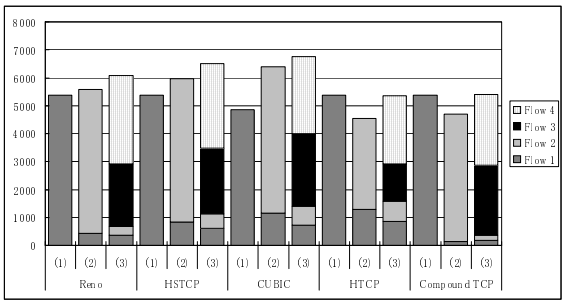

Figure 9: Time averaged throughputs in data transfer from two servers (different RTTs).

Figure 10 shows the time-series throughput characteristics of two coexisting flows when the socket buffer size was set to 100 MB and the resulting achievable throughput of a single flow was 5.4 Gbps. Flow 1 started transferring data first, and Flow 2 was established after 50 seconds.

In all cases, as in the cases in Fig. 8, Flow 1 was damaged when Flow 2 started. When HSTCP or HTCP is used, Flow 1's throughput recovers to the ssthresh and CUBIC flow recovers to its original level, while flow for Compound TCP

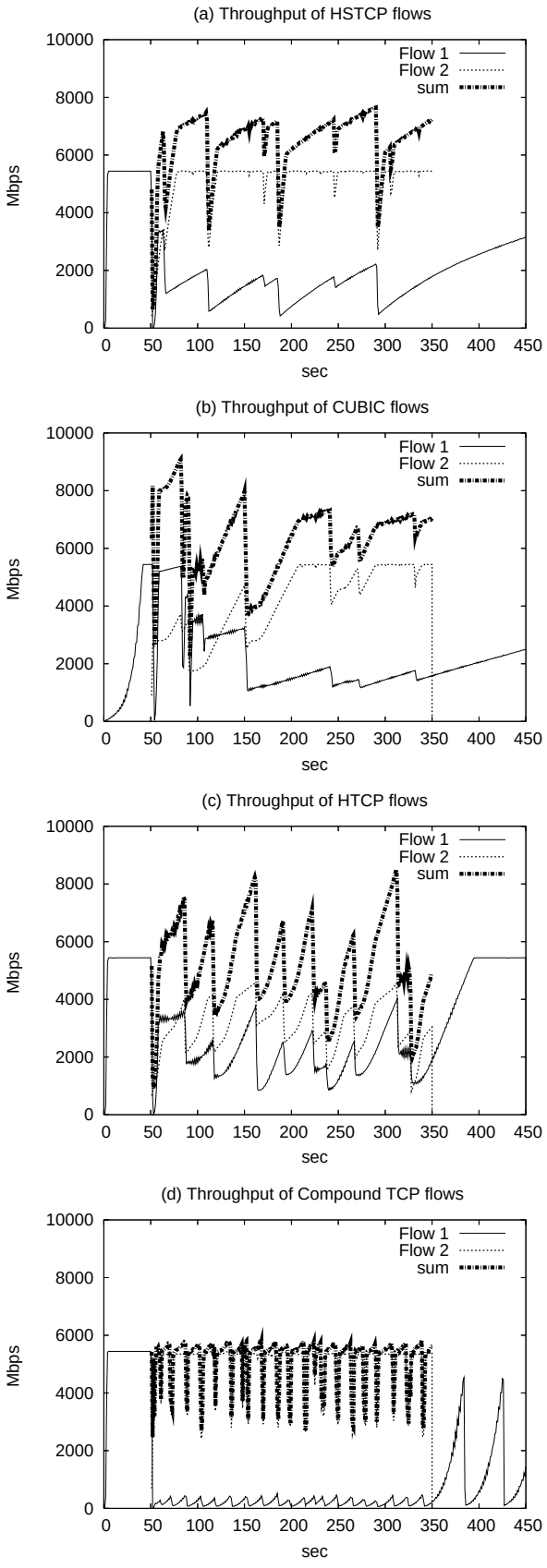

Figure 10: Data transfer from two servers in Fig.1 (b)

does not recover while Flow 2 is transferring data.

In the HSTCP protocol shown in Fig. 10(a), Flow 1's throughput quickly jumped to 5.4 Gbps, but Flow 2's throughput, which had the shorter RTT, jumped to 5.4 Gbps after it started and sustained this level. As a result, the throughput of Flow 1 decreased and then gradually started to increase. Then, the total throughputs of Flows 1 and 2 reached around 7 Gbps, i.e., their throughput decreased simultaneously. Then Flow 2 quickly recovered to 5.4 Gbps, while Flow 1 increased by degrees. For CUBIC (Fig. 10(b)), the throughput of Flow 1, which had a longer RTT, increased rapidly. Then, when Flow 2 started, Flow 1 decreased, as in HSTCP. For HTCP (Fig. 10(c)), we observed that the throughput of Flow 2 was slightly higher than that of Flow 1 while the throughputs of the two coexisting flows increased and decreased periodically. For Compound TCP, as shown in Fig. 10(d), after Flow 2, which had the shorter RTT, started, the throughput of Flow 1 did not increase. After Flow 2 stopped, the throughput of Flow 1 was able to recover to a level slightly lower than that observed before Flow 2 started.

We also investigated the same scenarios when the socket buffer size was adjusted to limit the maximum throughput of a single flow to 3 Gbps, as described in the previous subsection. We observed no differences in throughput characteristics from those observed in Fig. 10 for any protocol, including the Reno protocol.

3.5 Coexisting flows using standard TCP and high-speed transport protocol

On an end-to-end path with 1 Gbps or less bandwidth, the unfairness problem between a high-speed transport protocol and Standard TCP is well-known (e.g. [7]). We examined this problem where a high-speed transport protocol flow and a Standard TCP flow coexist on 10-Gbps shown in Fig. 1(a). We focused on two scenarios: (1) the high-speed transport protocol flow starts 50 seconds after the Standard TCP flow starts, and; (2) the high-speed transport protocol starts first. In both scenarios, the socket buffer sizes were set to either 40 MB or 100 MB.

The time-series in Scenario (1) with 100 MB socket buffer are shown in Fig. 11, where differences among the high-speed transport protocols were observed. Figure 11(a) shows the flow for CUBIC, whose throughput increased more slowly than that observed in Fig. 3(b).As shown in Fig. 11(b) (for HTCP) and (c) (for Compound TCP), the throughputs of those flows increased and decreased repeatedly. In all cases, after the high-speed transport protocol flow stopped transferring data, the Standard TCP could not recover quickly. In addition, the throughput behaviors remained unchanged even when the socket buffer size was changed.

In Scenario (2), based on experimental results not shown here, the high-speed transport protocol flow instantaneously decreased when Standard TCP flow starts its data transfer, however, the throughput recovered to its original level quickly. In contrast, the throughput of the Standard TCP flow increased very gradually; that is, the throughput of the Standard TCP flow was only several hundreds of Mbps at 100 s. This tendency was observed regardless of the type of high-speed transport protocols and the socket buffer size.

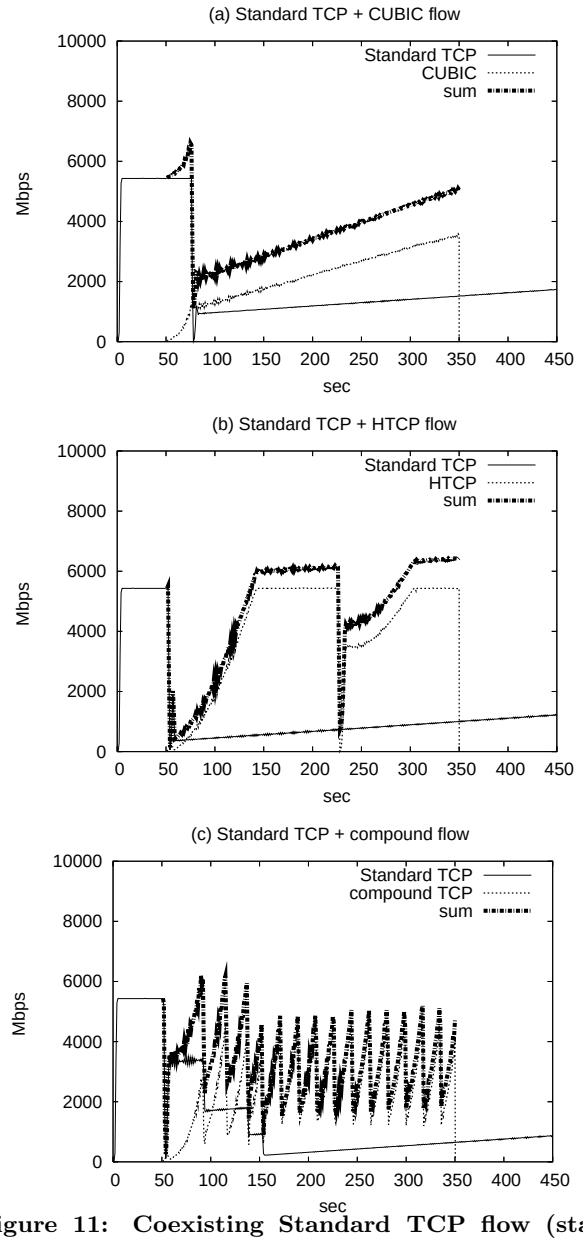

Figure 11: Coexisting Standard TCP flow (start first) and High-Speed TCP flow

4. CONCLUDING REMARKS

We investigated the performance of various high-speed transport protocols implemented in the Linux kernel through experiments on long distance 10-Gbps-class paths including US-Japan international lines, provided by JGN II, an open 10-Gbps-class network testbed in Japan.

We reported here the preliminary results of our experiments. We tested the TCP throughput of a single flow and then those of multiple coexisting flows on an end-to-end 10-Gbps path. We observed that for all the targeted high-speed transport protocols, a single flow could stably sustain the maximum throughput allowed by a bottleneck on the sender machine. When two individual flows coexisted, in most cases, the throughput of each flow fluctuated periodically, resulting in inefficient sharing of the network bandwidth. For example, imbalance in the time-averaged throughput between two flows would likely occur even if those flows have the same RTT, and the sum of those throughputs could not fully utilize the 10 Gbps bandwidth, especially when flows have the same RTT. We also observed that the performance of a high-speed transport protocol flow was damaged in throughput when a new TCP flow started after it. This is probably due to an aggressive slow-start mechanism harmful both to the coexisting flows and to itself. Therefore, a new gentle but effective start-up mechanism in high-speed transport protocols is one of the key issues in efficiently implementing coexisting multiple flows through the same bottleneck.

In future work, we are planning to examine a wider variety of network resource-sharing scenarios and to conduct more in-depth investigations (by monitoring TCP-internal or kernel-internal behaviors). We will also conduct experiments using the updated kernel version for various other high-speed transport protocols.

We give special thanks to Prof. Dirceu Cavendish for his helpful comments. We are grateful to the NCDM team members and the staff at APAN/JGNII NOC for their kind help in conducting our experiments. This work was supported in part by the JSPS, Grant-in-Aid for Scientific Research (S) (No. 18100001).

5. REFERENCES

[1] http://www.psc.edu/networking/projects/tcptune.
[2] S. Ha, Y. Kim, L. Le, I.Rhee, and L. Xu. A step toward realistic performance evaluation of high-speed TCP variants. In *PFLDnet2006*, February 2007.
[3] http://www.hamilton.ie/net/.
[4] G. Hisdel. 10gb ethenet back-to-back tests. June-August 2003.
[5] http://dast.nlanr.net/Projects/Iperf.
[6] C. Kost, S. McDonald, B. Caron, and W. Hong. End-to-end lightpaths for large file transfers over high speed long distance networks.
[7] K. Kumazoe, K. Kouyama, Y. Hori, M. Tsuru, and Y. Oie. Transport protocols for fast long-distance networks: Evaluation of their penetration and robustness on JGNII. In *PFLDnet2005*, Feb 2005.
[8] K. Kumazoe, K. Kouyama, Y. Hori, M. Tsuru, and Y. Oie. Can high-speed transport protocols be deployed on the internet? : Evaluation through experiments on JGNII. In *PFLDnet2006*, Feb 2006.
[9] K. Kumazoe, M. Tsuru, and Y. Oie. Investigating high-speed transport protocol flows on 10 gigabit testbed jgnii - with monitoring the linux kernel internally -. In *PACRIM2007*, Aug 2007.
[10] Y. Li, D. Leith, and R. N. Shorten. Experimental evaluation of TCP protocols for high-speed networks. *Transactions on Networking, 2006*.
[11] http://infocom.unirma1.it/ vacirca/year/cca.html.
[12] S.Ha, L.Le, I.Rhee, and L.Xu. Impact of background traffic on performance of high-speed TCP variant protocols. *Computer Networks, 2007*.
[13] http://www-didc.lbl.gov/TCP-tuning/.
[14] http://wil.cs.caltech.edu/.
[15] S. Ubik. Field trial with intel 10gigabit ethenet adapters for PC. Oct 2003.