

High-Speed TCP Performance Characterization under Various Operating Systems

Y. Iwanaga[†], K. Kumazoe[†], D. Cavendish[†], M. Tsuru[†] and Y. Oie[†]

[†]Kyushu Institute of Technology
680-4, Kawazu, Iizuka-shi, Fukuoka, 820-8502, Japan
yoichi@infonet.cse.kyutech.ac.jp, {kuma,dirceu,tsuru,oie}@ndrc.kyutech.ac.jp

ABSTRACT

Various TCPs for high-speed and long-distance networks have been implemented in Linux and Windows Operating Systems, being readily available to Internet users. However, their performance has been evaluated mainly through Linux servers and clients, despite the heterogeneity of OSs in the real Internet. In this report, the performance of high-speed transport protocol flows between servers and various client OSs is investigated in realistic Internet network scenarios.

Keywords: high-speed networks, high-speed transport protocols, Linux, Windows

1 Introduction

Modern OSs implement a variety of high-speed transport protocols to achieve efficient data transfer over large-scale and diverse networks. For example, the default transport protocol of Linux OS is now CUBIC[1], not Standard TCP(e.g. NewReno). Moreover, Linux implements various high-speed transport protocols, including HSTCP, HTCP and BIC. Users can select a protocol of choice for TCP flows via Command Line Interface. In addition, Windows OSs implement Compound TCP[2] from Vista release onwards.

So far many research groups have presented their performance evaluation experiments towards standardization of high-speed transport protocols. However, discussions on a methodology for performance evaluation across different transport protocols have not been very fruitful. For example, most of high-speed transport protocols, other than compound TCP, were implemented in Linux OS first due to the open source nature of Linux. Hence, performance evaluation of high-speed transport protocols have been conducted mainly via Linux servers and clients. As most clients adopt Windows OS, we believe it is important to investigate the impact of a variety of client OSs on the performance of TCP flows, in addition to the high-speed protocol run at the server OS.

In this report, we investigate the performance of TCP flows between servers and clients adopting Linux, Windows XP, Vista and Windows 7. Because we have no access to Windows code, we focus on characterizing the performance of Windows TCP stack, without a root cause analysis.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 describes the experimental environment used. Section 4 presents experimental results, and Section 5 presents a summary of our contributions.

2 Related work

HSTCP[3] was proposed in 2001 as a replacement to Standard TCP in order to tackle the problem of achieving efficient use of resources in large-scale and diverse networks. Since then, various high-speed transport protocols followed. Many research groups have studied these high-speed TCP protocols via simulation and experiments in testbed networks. Most of these activities have been conducted based on diverse performance measures and scenarios, making it difficult to compare performance of various transport protocol in a common environment. To improve this situation, an IETF draft [4] has been published to provide common network models, scenarios and performance measures.

Many research groups have conducted performance evaluation of high-speed transport protocol via experiments[5][6]. In these activities, target protocols were implemented in Linux first, so Linux servers and clients were adopted. In addition, we can find activities targeting MacOSX clients in [6] and targeting FreeBSD and XP as well as Linux[7]. However, these works deal with single flow characteristics only.

In this report, we investigate how the following two aspects impact on the performance of TCP flows in the Internet: (1) the type of protocols employed at the server (TCP data sender), (2) the type of OSs at the client (TCP data receiver).

3 Experimental Setup

The network topology used in our experiments is shown in Fig.1. We use the network emulator Hurricane II, from PacketStorm[8], so as to configure RTT flexibly in the range from 0[ms] to 230[ms]. In the following, we set path RTT to 180[ms], except when otherwise noted. This value represents an RTT between Japan and the US, and was selected as an example of a fast-long distance path. Two servers (server 1 and server 2) and two clients (client 1 and client 2) are used to establish TCP and UDP flows using iperf[9] between them. Although jumbo frames are supported in most Gigabit NICs, there is no guarantee that intermediate routers will support them. Hence, we show results for MTU size of 1500 bytes only. The buffer size at ingress and egress switches is 64 packets, the maximum size for the switches used. The server and receiver equipment specifications are listed in Table 1.

We target four client OSs, Linux and three Windows versions (Windows XP, Vista and Windows 7). The OS versions are shown in Table 1. We include old versions of Windows beside the latest Windows 7, because legacy machines are likely to coexist with new ones in the Internet.

As for each OS, we perform parameter tuning to achieve high performance based on various technical information such as [10].

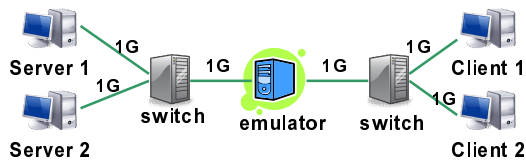


Figure 1: Network Configuration

Table 1: Equipment specifications

	Server	Client
OS	Fedora Linux (2.6.25)	Fedora Linux (2.6.29.4) Windows XP Pro. x64 SP2 Vista Ultimate 64 bit SP2 Windows 7 Ultimate 64 bit
CPU	Xeon 3.33 GHz	Xeon 2.4GHz
Memory	6 GByte	2 GBytes
PCI bus		64 bits
NIC		e1000

Servers run Linux OS, which implements various high-speed transport protocols natively except for Compound TCP. Among various high-speed transport protocols, we experiment with Reno, CUBIC and Compound TCP in our experiments as most TCP representatives. CUBIC is the current default TCP algorithm in Linux and Compound TCP is implemented in Windows (Vista, Windows 7 and Windows Server 2008). We use the patch code for running Compound TCP on Linux instead of adopting Windows Server 2008 at the server. This patch was developed by researchers of Caltech and Microsoft, hence it is likely that the implemented Compound TCP be very similar to that implemented in Windows. This patch works on Kernel 2.6.25, being used in the server side. The latest Kernel version is used at the client side in our experiments.

By monitoring SYN packets at the beginning of the TCP session, we verify that all targeted OSs enables options such as windows scale, Timestamp and SACK.

In order to check the status of the session path, we establish a single UDP flow of 900[Mbps] between servers and clients and verify that there are no losses on the connections for all target OSs before running the experiments. From this observation, we can say the path status including end hosts in the experimental setup shown in Fig.1 is not lossy. In the following sections, we run experiments over clean paths.

4 Experimental Results

We show the throughput characteristics of a single flow adopting each high-speed transport protocol in subsection 4.1 and those of two coexisting flows in subsections 4.2-4.4. Performance measures are: throughput characteristics, measured via iperf; TCP related parameters, monitored by web100[11]. The duration of each trial is 100 [sec] in the single flow sce-

nario, while it is 300[sec] in the coexisting flows one. Results are reported as averages over five trials.

4.1 Single flow characterization

In this section, we show the throughput characteristics of a single flow for various OS clients. First we show the throughput characteristics in a path for various RTTs.

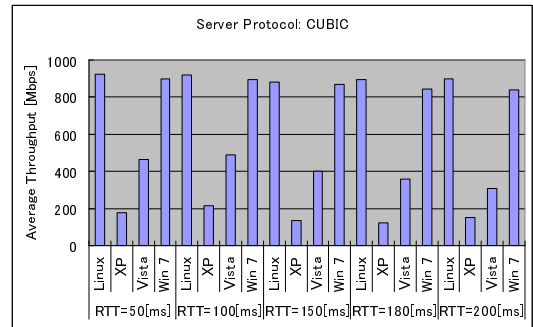


Figure 2: Averaged Throughput for a Single Flow; Server Protocol: CUBIC; Client OS: Linux, XP, Vista, Windows 7

Figure 2 shows throughput characterization of Flow 1, established between server 1 and client 1 in Fig.1 with CUBIC as the server high-speed transport protocol. In this scenario, we set RTT along the path to 50, 100, 150, 180 and 200[ms]. Figure 2 shows that throughput characteristics of Linux and Windows 7 clients are not affected by different values of RTT, with stable average throughput around 900[Mbps]. The throughput of XP client, on the other hand, is limited at around 200 [Mbps] for all RTTs. The characteristics of Vista client achieves about 1.5 to 2 times of that of XP client on average. From these results, it is obvious that clients experience different throughputs, depending on the kind of OSs. From this result, we can say that Linux and Windows 7 clients can download files from Linux server over various RTT paths with stable performance in a single flow.

Next we investigate the throughput performance of sessions at path conditions that vary over time. In Fig 1, after 100 [s] of a single high speed TCP flow is established between server 1 and client 1, a UDP flow of 200[Mbps], that is 20 percent of the bottleneck link, starts data transfer for the duration of 100 [s] between server 2 and client 2. In addition, a single TCP flow keeps its data transfer for an additional 100 [s] after the UDP flow stops its data transfer. Figures 3 and 4 show the average throughput characteristics with UDP flow as cross traffic on the data transfer from Linux (CUBIC protocol) and Windows (Compound TCP) servers.

In the cases of running CUBIC at the server, we observe that the average throughput of a single flow (0-100[s]) is around 900 [Mbps] when the receiver OSs are Linux and Windows 7. After UDP flow starts, throughput degrades drastically for the flows running Linux and Windows 7 at the receivers. After UDP flow terminates, at 200 [s], sometimes (two or three of five trials) Linux and Vista clients recover their throughput level to 900[Mbps]. For Compound TCP protocol at the server, Linux, Vista and Windows 7 clients sometimes (e.g.

several times of 5 trials) recover their throughput after UDP flow exits.

When comparing the throughput performance of CUBIC and Compound TCP servers, we observe a sharp difference in behavior of their throughput recovery phases. Windows 7 client is sometimes able to recover its throughput level after the UDP flow is gone when the server is running Compound TCP, while the client is not able to increase throughput again when the server is running CUBIC. For all protocols, XP and Vista clients show poor performance, similar to Fig.2 throughput performance.

We can rank the packet losses observed in UDP flow in the following order, from higher to lower: Linux, Windows 7, Vista, and XP. This is due to poor performance of XP/Vista, these clients do not experience congestion when sharing the bottleneck with UDP traffic.

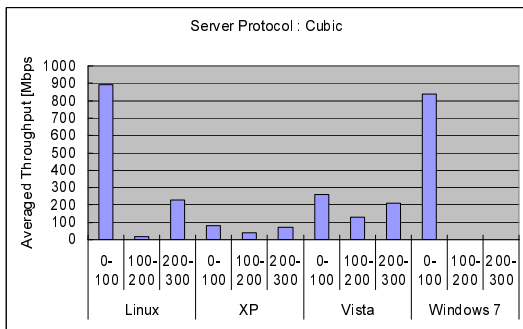


Figure 3: Coexisting UDP flow (100[s]-200[s]); Server Protocol: CUBIC; Client OSs: Linux, XP, Vista, Windows 7

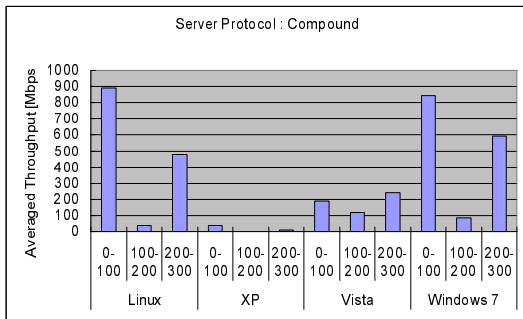


Figure 4: Coexisting UDP flow (100[s]-200[s]); Server Protocol: Compound; Client OSs: Linux,XP,Vista,Windows 7

We further look into TCP parameters to investigate the performance behavior of the various protocols. In the following, we show TCP related parameters monitored via web100 on the flows adopting various protocols and client OSs in the coexisting UDP scenario during 100 to 200 [s].

Figure 5 shows the cwnd when the server adopts Reno protocol. Linux opens its cwnd quickly and keeps it high until UDP traffic starts, at which point it reduces the cwnd. Other clients show similar behavior at the beginning of the flow but Vista and XP clients shrink their cwnd value even before UDP flow starts. Windows 7 can keep the high cwnd value similar to the case of Linux. When the UDP flow starts at 100 [s], all clients have similar behavior in time in cwnd, that is, they shrink cwnd and never increase again.

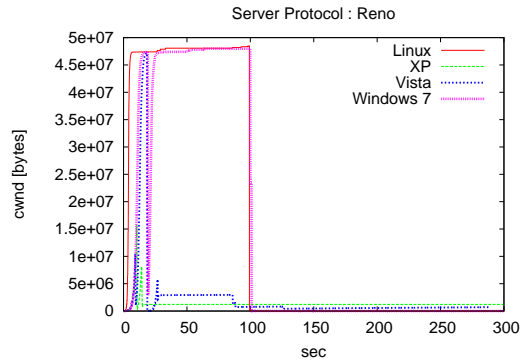


Figure 5: cwnd, Server Protocol: Reno

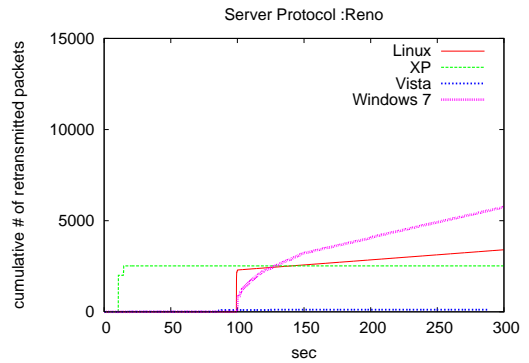


Figure 6: # of cumulative retransmitted packets, Server Protocol: Reno

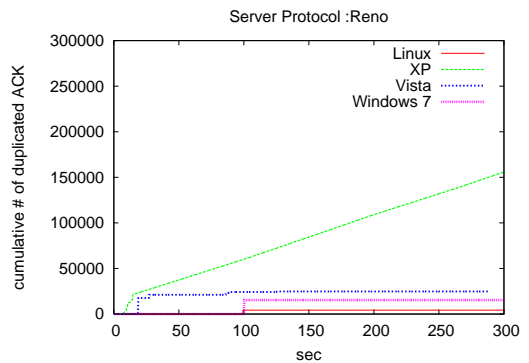


Figure 7: # of cumulative duplicated packets, Server Protocol: Reno

Figure 6 - 8 present cumulative # of retransmitted packets, duplicated Acknowledgement packets and timeout adopting Reno at the server. We observe a large number of packet retransmissions following duplicate ACK packets for Linux and Windows 7 clients when the UDP flow starts its data transfer. After the UDP flow exits, the number of retransmitted packets continues to increase for Linux and Windows 7 clients. This means that clients have not been able to retransmit all packets that got lost during UDP traffic, thus, the cwnd never opens up. As for XP client, timeout, duplicate ACK and packet retransmissions were observed at 10[s], where the network is

not congested yet. After that, the # of duplicate ACK packets increases but the # of retransmitted packets are flat. Thus, XP client is not able to recover its cwnd again. Also, Vista client starts to generate duplicate ACKs at 20[s], although the network is not congested at that time.

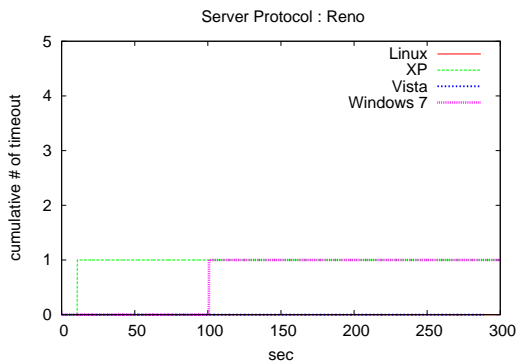


Figure 8: timeout, Server Protocol: Reno

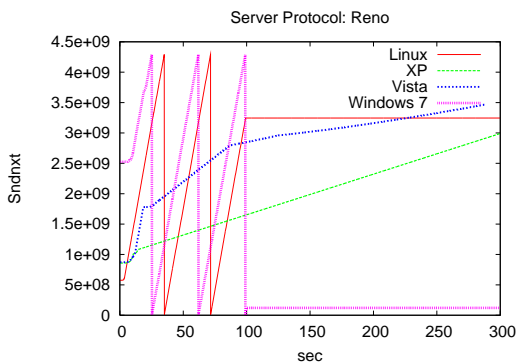


Figure 9: sndnxt, Server Protocol: Reno

Figure 9 shows the sndnxt pointer (the sequence number of the next byte of data to be sent[12]) supported in the Linux kernel, monitored via web100 at the server side. As the behavior of sndnxt pointer at the server is driven by ACK packets from the client, we can trace the behavior of the ACK packets generated by the client by monitoring the sndnxt pointer. As the counter range of sndnxt is between 0 and 2^{32} , the graph shows a max sndnxt value of $2^{32} - 1$ wrapping up to 0. During 0 -100 [s], we observe that the increase rate of sndnxt of Linux and Windows 7 clients are identical, while the increase rate between Linux client and Windows clients are different, which might be caused by different implementations of ACK packet generation at each OS.

Figure 10 shows the cwnd when the server adopts CUBIC. Comparing with Reno in Fig.5, CUBIC is more aggressive in opening up its cwnd after UDP traffic is gone. Linux and Vista clients are able to increase their cwnd after UDP flow exits, while XP and Windows 7 clients are not able to recover their cwnd. Figure 11 and 12 present the cumulative # of retransmitted and duplicate ACK packets. UDP triggers too many retransmissions, causing slow recovery for all protocols. We observe that CUBIC at the server is causing a larger

number of duplicate ACKs and retransmissions than Reno as shown in Fig 6 and 7. CUBIC tries to open up its cwnd more quickly, even in the presence of many retransmissions. For XP and Vista clients, we observe packet retransmissions at the beginning of the flow even without UDP flow, when the server protocol is CUBIC, similarly to Reno. As shown in Fig.13, UDP flow causes timeout for Linux and Windows 7 client. However, Linux clients recovers their cwnd when UDP traffic exits, while Windows 7 can not recover its level.

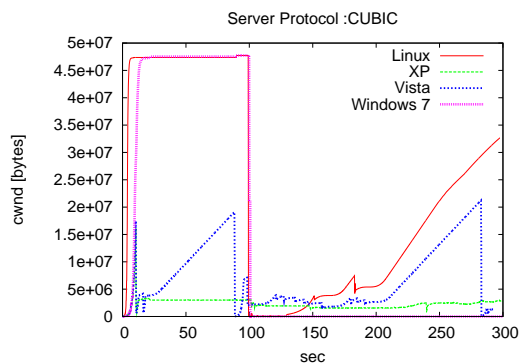


Figure 10: cwnd, Server Protocol: CUBIC

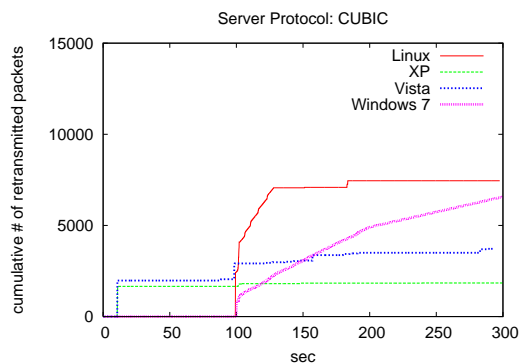


Figure 11: # of cumulative retransmitted packets, Server Protocol: CUBIC

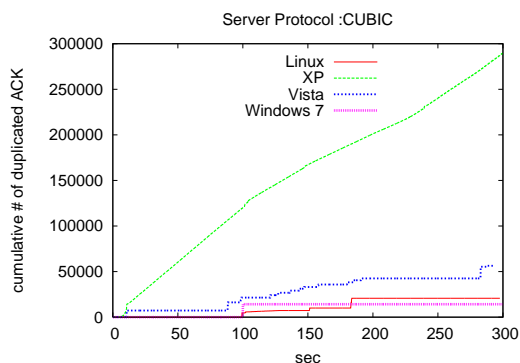


Figure 12: # of cumulative duplicated packets, Server Protocol: CUBIC

Figure 14 shows the cwnd when the server adopts Compound TCP. Linux and Windows 7 clients have very good

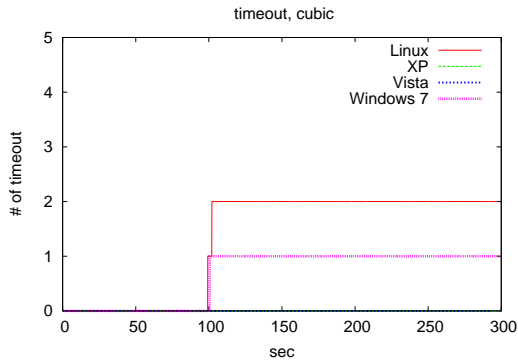


Figure 13: timeout, Server Protocol: CUBIC

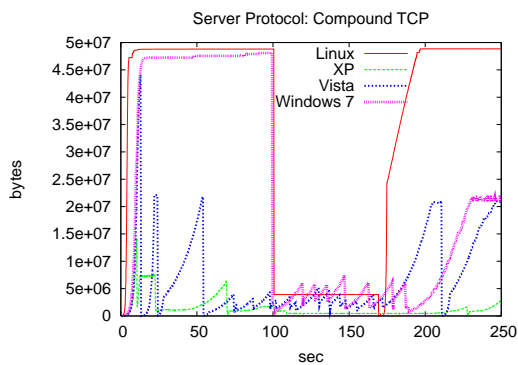


Figure 14: cwnd, Server Protocol: Compound

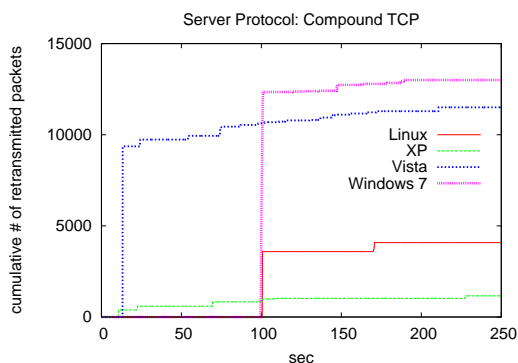


Figure 15: # of cumulative retransmitted packets, Server Protocol: Compound

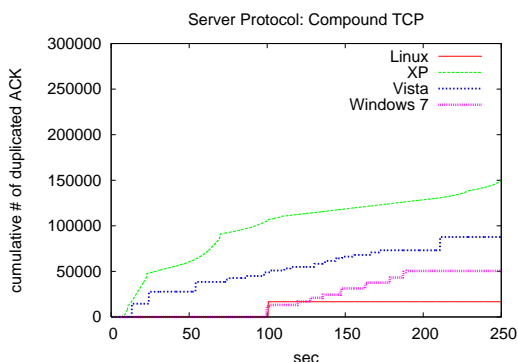


Figure 16: # of cumulative duplicated packets, Server Protocol: Compound

cwnd behavior. That is, their cwnd reduces only when UDP traffic is active, while XP and Vista clients do not open their cwnd enough. UDP flow causes timeout for Linux client. However, its clients recovers their cwnd when UDP traffic exits. After UDP traffic is gone, the number of retransmitted packets remain flat for Linux and Windows 7 clients as shown in Fig. 15, which explains Linux and Windows 7 client cwnd recovery after UDP flow is gone. XP and Vista clients start to generate duplicate ACK packets even before UDP flow enters the system as shown in Fig. 16, which is a similar behavior as the case of the server running Reno and CUBIC. Moreover, duplicate ACKs continue to increase after UDP traffic exist the system.

From these observations, we can conclude the following about the behavior of client OSs and protocols at the server.

- The increase of duplicate ACKs even before UDP traffic start interfering with TCP sessions between servers and XP or Vista clients indicates that there might be some problems in clients' delayed ACK mechanisms, which is independent of the protocols run at the server.
- CUBIC and Compound TCP are able to open their cwnd depending on the status of the network. However, Windows 7 client is not able to recover its throughput after the UDP flow exits when the server runs CUBIC (Fig.10), whereas it recovers its throughput when the server runs Compound TCP (Fig.14). Comparing the number of retransmitted packets observed in Windows 7 clients in those cases (Fig.11 and 15), we observe that a larger number of retransmitted packets occurred when the server runs Compound TCP, although the number of retransmissions did not increase after the UDP flow is gone. So, the flow is able to recover in case of the server running Compound TCP. On the other hand, when the server protocol is CUBIC, although the number of retransmitted packets at UDP flow start time is smaller than that of the server running Compound TCP, it increases after the UDP flow exits, hence the flow might not be able to reopen its cwnd back to its previous level.

4.2 Characteristics of coexisting intra-protocol flows

In this subsection, we examine the throughput characteristics of multiple coexisting flows using a same transport protocol in scenarios where flows run from server 1 to client 1 and from server 2 to client 2 with the same RTT of 180 [ms]. The bottleneck is located at the ingress switch close to the senders in Fig.1.

Figure 17 shows the total throughput of two flows for Linux clients and CUBIC servers. The flows are simultaneously established between the server - client pairs (server 1 - client 1, server 2 - client 2) on a path with RTTs of 50, 100, 150, 180 and 200 [ms], respectively. We can see that the total throughput of both flows is affected by the RTT experienced. That is, the larger the RTT, the smaller the throughput of each flow and the total throughput of two flows. In our configuration,

the total throughput of the two flows is about 80 percent of the bottleneck link when RTT is 50[ms], while it is 40 percent on the path when RTT is 200[ms]. From these results, we can see that achieving RTT fairness among TCP flows of various RTTs is problematic.

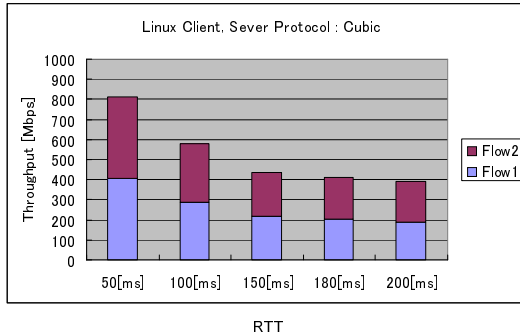


Figure 17: Time averaged throughputs for various path RTT; Server Protocol: CUBIC; Client OS: Linux

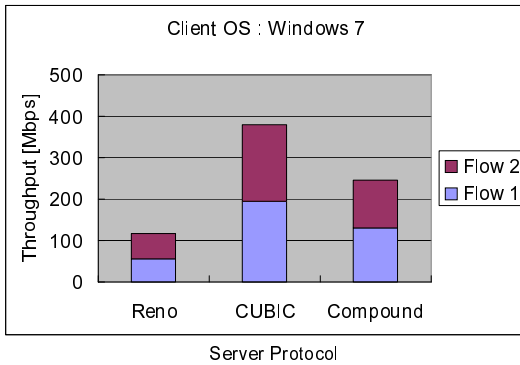


Figure 18: Average Throughput for various Server Protocols

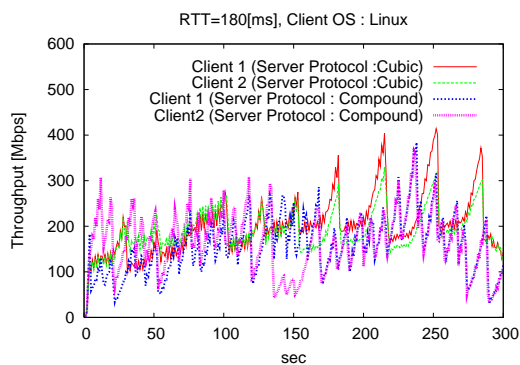


Figure 19: Time series throughput, Client OS: Linux; Server Protocol: CUBIC and Compound

We also investigate the total throughput of two coexisting flows for each client OS. Two flows, Flow 1 and Flow 2, start their transfer simultaneously. We also have run the scenario where Flow 2 starts its data transfer two seconds after flow 1, and found the tendencies on the performance similar to the case of starting simultaneously. Thus, only the performance

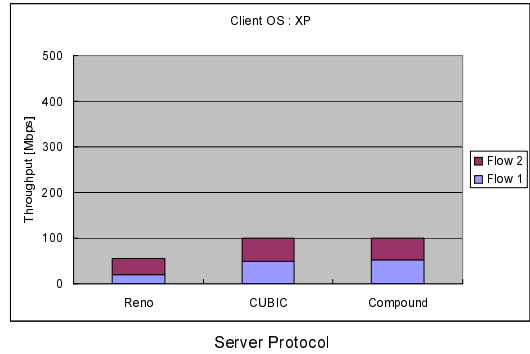


Figure 20: Average Throughput for various Server Protocols

of starting two flows simultaneously is presented for sake of brevity.

First, when servers use Reno protocol, the throughput of Windows 7 clients is very limited, as shown in Fig.18, around 50 [Mbps] for each client, with total throughput of about 100 [Mbps]. When Linux servers adopt CUBIC protocol and two Windows 7 clients download files simultaneously, the total average throughput is about 400 [Mbps]. When the server is Windows running Compound TCP, the total throughput of the two flows is about 250 [Mbps].

As for Linux client, similar tendencies are observed as of those of Windows 7 client. Fig.19 shows the time-series throughput characteristics of Linux client for CUBIC and Compound TCP flows. We observe that two coexisting flows increase and decrease their throughput repeatedly, with the range of variation of Compound TCP flows being a little larger than that of CUBIC flows. It can be said that this difference of the two protocol behaviors causes the difference on average throughput performance between the two protocols.

In case of XP and Vista clients, the total throughput of the two flows is very limited. XP clients' throughput is about 100 [Mbps] for CUBIC and Compound TCP protocols, as shown in Fig.20.

Vista clients' throughput is about twice that of XP. Similar to the single flow scenario, throughput of XP and Vista clients is very limited, even though their paths do not experience any network congestion. Hence, in what follows, we investigate whether the limited throughput might be due to losses in the internal OSs itself, by characterizing Linux and Windows 7 clients.

4.3 Standard TCP and High-Speed transport protocols

On end-to-end paths with 1 [Gbps] or less bandwidth, unfairness problems between a high-speed transport protocol and Standard TCP is well-known in environments of Linux clients[7]. In this section, we examine the fairness issue of XP, Vista and Windows 7 OS clients. We focus on two scenarios : (1) high-speed transport protocol flow starts 2 seconds after a Standard TCP flow starts (case 1), and : (2) high-speed transport protocol starts earlier than a Standard TCP flow (case 2).

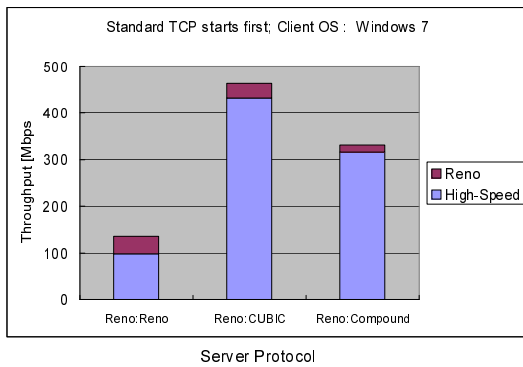


Figure 21: Coexisting Standard TCP and High-speed TCP flows; Client OS: Windows 7

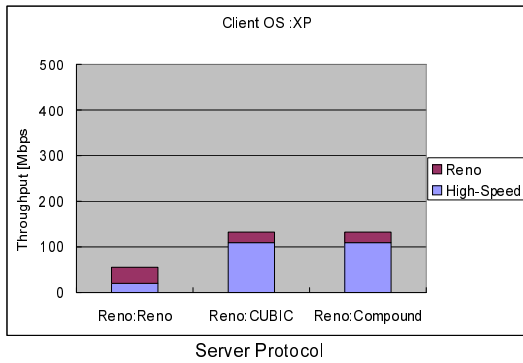


Figure 22: Coexisting Standard TCP and High-speed TCP flows; Client OS: XP

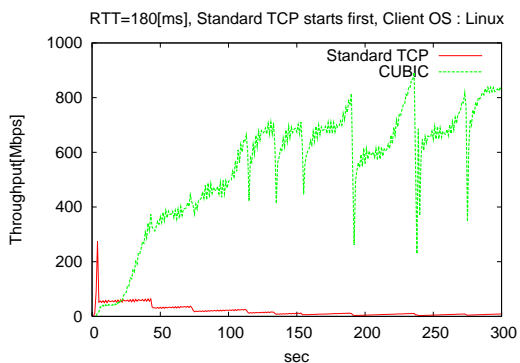


Figure 23: Time-series Throughput of coexisting Standard TCP and CUBIC Servers, Client OS: Linux

Figure 21 shows average throughput in scenarios of Windows 7 clients for case 1. When both servers adopt Standard TCP, the total throughput of two flows is about 120[Mbps]. In case 1, where the Standard TCP flow starts first, for both cases of CUBIC and Compound TCP protocol at the servers, the total throughput increases as compared with two coexisting standard (Reno) TCP flows. However, the throughput of Standard TCP flow is drastically less than that observed in coexisting two standard TCP flows. The tendencies observed in case 2 are similar to case 1. In examining the case of two flows starting simultaneously, we observe similar tendencies, that is, the high-speed transport protocol flow affects the performance of Standard TCP flow. In addition, comparing the performance of the scenario where two flows coexist in the shorter RTT path (50 [ms]) with longer RTT path (180[ms]), the total throughput of two flows increased. Therefore, we can say that high link utilization is achieved on scenarios where standard TCP and high-speed transport protocol flows coexist in short RTT paths. However, the throughput of Standard (Reno) TCP flow is also affected by the high-speed transport protocol flow for both CUBIC and Compound TCP server protocols.

For XP clients, the average throughput for a client using high-speed transport protocol is higher than clients using Standard TCP as shown in Fig.22. The total throughput of two coexisting flows is limited to 100[Mbps] for all protocols. Vista clients show the same tendencies as those of XP, with the difference that the total throughput is around 250[Mbps]-300[Mbps] when the server runs CUBIC and Compound TCP and 150[Mbps] when servers run Standard TCP. Linux client shows almost the same tendencies of that of clients running Windows 7.

Figure 23 shows the time-series throughput characteristics of two coexisting flows, whose servers run Standard TCP and CUBIC, for Linux clients, when the server running Standard TCP starts transferring data first. We can observe that the performance of high-speed transport protocol flow overwhelms that of the Standard TCP flow, even though the later starts data transfer first. All client OSs (Linux, Windows XP, Vista and Windows 7) for which data was transferred via servers running Standard TCP are negatively affected by the high-speed transport protocol flows.

4.4 Coexisting client OSs

So far, we have shown the characteristics of multiple flows in configurations where the clients adopt identical OS, e.g. Client 1- Client 2 is Linux - Linux, XP - XP, Vista - Vista and Windows7 - Windows7. In this subsection, we focus on scenarios where different client OSs coexist in the configuration of Fig. 1. Coexisting flows start their data transfer simultaneously and lasts 300 [s].

Table 2 - 4 summarize the average throughput of two coexisting flows (Client 1 / Client 2) for each protocol at the server.

Table 2: Average Throughput (Server Protocol: Reno)

Client 1 \ Client 2		Client 2			
		XP	Vista	Windows 7	Linux
XP	19.8 / 36.2	9.5 / 99.8	27.7 / 205.7	5.6 / 230.1	
Vista	99.8 / 9.5	62.6 / 51.4	42.8 / 67.1	15.4 / 307.3	
Windows 7	205.7 / 27.7	67.1 / 42.8	55.6 / 62.1	15.9 / 217.2	
Linux	230.1 / 5.6	307.3 / 15.4	217.2 / 15.9	55.2 / 63.1	

Table 3: Average Throughput (Server Protocol: Cubic)

Client 1 \ Client 2		Client 2			
		XP	Vista	Windows 7	Linux
XP	50.0 / 50.1	48.1 / 135.5	64.2 / 232.6	48.4 / 325.8	
Vista	135.5 / 48.1	70.3 / 90.1	80.7 / 96.1	140.4 / 259.3	
Windows 7	232.6 / 64.2	96.1 / 80.7	195.4 / 184.3	228.9 / 267.0	
Linux	325.8 / 48.4	259.3 / 140.4	267.0 / 228.9	205.4 / 205.2	

Table 4: Average Throughput (Server Protocol: Compound)

client 1 \ client 2		client 2			
		XP	Vista	Windows 7	Linux
XP	53.0 / 46.6	6.2 / 258.4	6.1 / 341.6	3.3 / 463.1	
Vista	258.4 / 6.2	94.3 / 97.8	86.6 / 119.5	87.6 / 460.9	
Windows 7	341.6 / 6.1	119.5 / 86.6	131.2 / 114.4	102.5 / 221.8	
Linux	463.1 / 3.3	460.9 / 87.6	221.8 / 102.5	153.6 / 147.6	

In what follows, a summary of average throughput in each scenario is shown:

- XP client and other OS clients:** In all cases, XP client can not increase its cwnd as much as the coexisting client. The average throughput is limited to around 10-50 [Mbps] for each protocol, while Vista client achieves 100-300 [Mbps], Window 7 client achieves 200-350 [Mbps], and Linux client reaches 200-450[Mbps].
- Vista client and other OS clients:** The performance of Vista client achieves higher throughput than a coexisting XP client. When Vista and Linux clients coexists, Linux client achieves higher throughput than that of Vista client. When adopting CUBIC protocol at the server, throughput is 260 [Mbps] for the Linux client and can download data from the server two times faster than Vista client and five times faster (throughput is 460 [Mbps]) when adopting Compound TCP.
- Windows 7 client and other OS clients:** Windows 7 client is able to download data from Linux server running CUBIC about two or three times faster than that of the XP client. For the Windows server running Compound TCP, the performance for XP client seems very limited, while Windows 7 client reaches over 200 - 300 [Mbps] throughput. In case of coexisting Windows 7 and Vista clients, the throughput of each client is almost the same when using Linux server (CUBIC) and Windows 7 (Compound TCP). That is, the throughput of each flow is lower than 120[Mbps]. Hence when

Windows 7 and Vista clients download data simultaneously, the total throughput is very limited, around 200 [Mbps].

- Linux client and other OS clients** When coexisting Linux and Windows clients download data from the servers simultaneously, Linux client gets higher throughput than those of Windows OSs. Figure 24 shows one of the examples, with coexisting Windows 7 and Linux Client. Both Linux and Windows 7 clients have almost identical performance when they download data from the Linux server. However, when downloading data from Windows server running Compound TCP, Linux client gets data about two times faster than Windows 7 client. In addition, when the Windows server runs Reno, the performance of Windows 7 client is very limited.

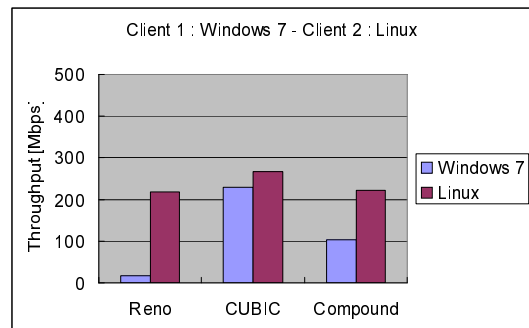


Figure 24: Coexisting Different OS Clients: Windows 7 - Linux

Figures 25 - 26 show the cwnd of coexisting Windows 7 and Linux clients. From these figures, when downloading data from the Linux server running CUBIC, both clients have similar performance. When data is downloaded from the Windows server using Compound TCP, Linux client starts up faster than Windows 7 client and keeps higher cwnd level than that of Windows 7 clients for a while. Then, both of them increase and decrease their cwnd simultaneously at almost the same level. The difference of throughput characteristics observed between Linux and Windows 7 clients in Table 4 comes from the superiority of the Linux client at the beginning of the flow.

From these results, we can rank the high-speed data transfer throughput protocols in the following order, from higher to lower: Linux, Windows 7, Vista, and XP.

5 Summary

Several high-speed transport protocols have been proposed and implemented in various kinds of OSs, aiming at more efficient communication in the Internet. In this report, experimental results of high-speed transport protocol conducted in an environment emulating the Internet were presented. Here is what we have learned:

- Both transport protocols at the server and client OSs affect performance. For example, the timing of ACK

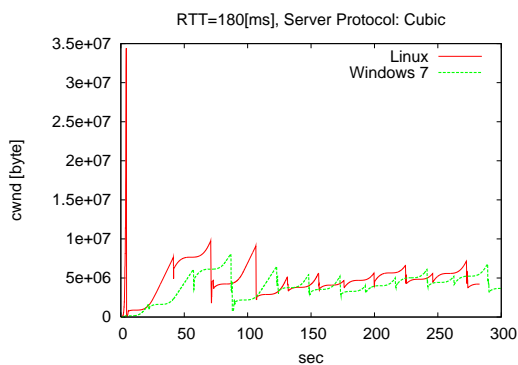


Figure 25: Coexisting Windows 7 and Linux Clients; Server Protocol: CUBIC

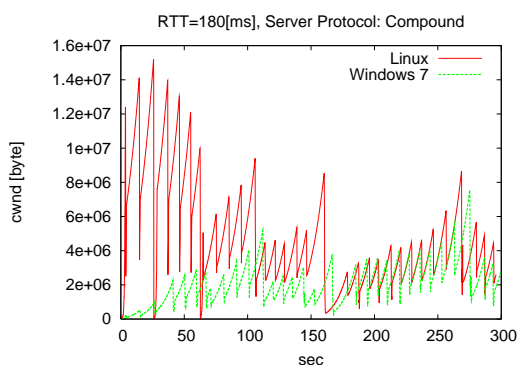


Figure 26: Coexisting Windows 7 and Linux Clients; Server Protocol: Compound TCP

packet generation affects the performance of TCP flows. This is because the congestion control and adjustment of sending rate are driven by information in the ACK packet stream. Thus, it is important to consider not only the algorithm of protocol adopted at the server but also the implementation of the client OS when analyzing the performance of TCP flows in the Internet.

- In specific scenarios, similar performances have been observed for Linux and Windows 7 clients. In case of coexisting Linux and Windows 7 clients, Linux server running CUBIC protocol provides clients with similar performance. On Windows server running Compound TCP, Linux clients have better performance at its start up phase than Windows clients.
- Comparing Compound TCP running on Windows server and CUBIC running on Linux server, in our configurations, CUBIC shows stable performance in throughput for most of the scenarios studied.

As future work, we are planning to examine a wider variety of network resource-sharing scenarios. We are also planning to investigate throughput performance in wired-wireless network environments. Also, we are planning to run experiments with Windows Server 2008.

REFERENCES

- [1] Rhee, L. Xu and S. Ha, CUBIC for Fast Long-Distance Networks, IETF, Internet Draft, draft-rhee-tcpm-cubic-02.txt(2007).
- [2] M. Sridharan, K. Tan, D. Bansal, D. Thaler, Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Network, IETF, Internet Draft, draft-sridharan-tcpm-ctcp-00.txt(2008).
- [3] S. Floyd HighSpeed TCP for Large Congestion Windows, RFC 3649, Experimental(2003).
- [4] L. Andrew, S. Floyd and G. Wang, Common TCP Evaluation Suite, draft-irtf-tmrg-tests-02.txt(2009).
- [5] <http://wil.cs.caltech.edu/>
- [6] http://netsrv.csc.ncsu.edu/wiki/index.php/TCP_Testing
- [7] 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, PFLDnet 2006(2006).
- [8] <http://www.packetstorm.com/psc/psc.nsf/site/Hurricane-II-software>
- [9] <http://sourceforge.net/projects/iperf/>
- [10] <http://www.psc.edu/networking/projects/tcptune/>
- [11] <http://www.web100.org/>
- [12] http://www.tcpiptide.com/free/t_TCPslidingWindowDataTransferandAcknowledgementMech-2.htm