A Transport-Layer Approach for Improving Thin-Client Performance in a WAN Environment

Yukio Ogawa*

Yokohama Research Laboratory, Hitachi, Ltd.,
292 Yoshida-cho, Totsuka-ku, Yokohama-shi, Kanagawa-ken, 244–0817 Japan
E-mail: yukio.ogawa.xq@hitachi.com
*Corresponding author

Go Hasegawa and Masayuki Murata

Graduate School of Information Science and Technology, Osaka University,
1–5, Yamadaoka, Suita-shi, Osaka-fu, 565–0871 Japan
E-mail: hasegawa@cmc.osaka-u.ac.jp
E-mail: murata@ist.osaka-u.ac.jp

Abstract: The performance of thin-client systems based on TCP depends on network quality, so it becomes worse in a WAN environment; however, the effects of TCP mechanisms have not been clarified. In this paper, we first describe the download traffic of thin-client systems as a two-state model with interactive data flows in response to keystrokes and bulk data flows related to screen updates. Since users are more sensitive to the keystroke response time, our next objective is to minimise the latency of interactive data flows, especially when the network is congested. Through detailed simulation experiments, we reveal that the main delays are queuing delay in the bottleneck router and buffering delay in the server. We then enhance two TCP mechanisms: retransmission timeout calculation and SACK control, which negate the negative impacts of existing options and increase the interval between occurrences of large delays by about four times.

Keywords: thin client; performance; WAN; wide area network; transport layer; interactive data flows; bulk data flows; retransmission timeout; TCP SACK.

Reference to this paper should be made as follows: Ogawa, Y., Hasegawa, G. and Murata, M. (2011) 'A Transport-Layer Approach for Improving Thin-Client Performance in a WAN Environment', *Int. J. Internet Protocol Technology*, Vol. 1, Nos. 3/4, pp.1–10.

Biographical notes: Yukio Ogawa received the M.S. degree in Science from Nagoya University and joined the Hitachi Central Research Laboratory in Japan in 1994. He is currently a senior researcher in the Hitachi Yokohama Research Laboratory in Japan. His research interests include network architectures for enterprise systems. He is a member of IEEE and IEICE.

Go Hasegawa received the M.E. and D.E. degrees in Information and Computer Sciences from Osaka University, Osaka, Japan, in 1997 and 2000, respectively. From July 1997 to June 2000, he was a Research Assistant in the Graduate School of Economics, Osaka University. He is now an Associate Professor at the Cybermedia Center, Osaka University. His research is in the area of transport architecture for future high-speed networks and overlay networks. He is a member of IEEE and IEICE.

Masayuki Murata received the M.E. and D.E. degrees in Information and Computer Science from Osaka University, Japan, in 1984 and 1988, respectively. In April 1984, he joined the Tokyo Research Laboratory, IBM Japan, as a Researcher. From September 1987 to January 1989, he was an Assistant Professor with the Computation Center, Osaka University. In February 1989, he moved to the Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University. In April 1999, he became a Professor at the Cybermedia Center, Osaka University, and since April 2004, he has been with the Graduate School of Information Science and Technology, Osaka University. He has published more than five hundred papers in international and domestic journals and conferences. His research interests include computer communication network architecture, performance modelling, and evaluation. He is a member of IEEE, ACM, and IEICE.

1 Introduction

A thin-client system is a server-centric computing system in which a client terminal transmits user inputs (keystrokes, mouse clicks, etc.) to a remote server, and the server returns the corresponding screen updates to the desktop application interface on the terminal (Figure 1). This system enables an enterprise's IT department to manage its client computing resources in an integrated fashion and to promote flexibility in the workplace without leaking corporate information from client terminals (Sagawa and Koda, 2009).

The thin-client system is generally implemented by using a remote desktop protocol such as X11 (Scheifler and Gettys, 1997), Virtual Network Computing (VNC®) (Richardson et al., 1998), or Microsoft® Remote Desktop Protocol (RDP) (MSDN®, 2011). From a transport-layer perspective, each remote desktop protocol is based on a persistent Transmission Control Protocol (TCP) connection. The system performance is thus affected by the TCP configuration. For example, the use of TCP buffering mechanisms—the Nagle algorithm (Nagle, 1984) and delayed acknowledgment (Braden, 1989)—could lead to delayed delivery of small packets in an interactive application (Minshall et al., 1999; Mogul and Minshall, 2001). Disabling these buffering mechanisms is therefore appropriate for an application like X11 (Stevens, 1994). Furthermore, TCP's slow-start restart (Jacobson and Karels, 1988), i.e., reinitialisation of the TCP congestion window (*cwnd*) after idle periods, even when there are no dropped packets, loses the benefit of a persistent connection and reduces the transmission rate (Heidemann, 1997; Visweswaraiah and Heidemannk, 1997).

The performance of a thin-client system from the user's point of view under various network conditions has been measured for typical desktop applications like a word processor or presentation creator (Nieh et al., 2003; Schlosser et al., 2007). Furthermore, network latency has been identified as a key factor determining the performance, especially in a wide area network (WAN) environment (Lai and Nieh, 2006; Tolia et al., 2006). According to McCabe (2003), network delay is the predominant delay for an interactive application like Telnet, and this type of application is sensitive to response time. We have identified that such delay requirements should be met for thin-client systems because users are very sensitive to delays and jitter in the response packets when they type on a thin-client terminal in a remote office. Although it has been shown that the system response time becomes worse in a WAN environment, the effects of TCP mechanisms have not been clarified.

We have therefore been investigating this subject. We previously described the characteristics of traffic (Ogawa et al., 2007) and then outlined an approach for improving the interactive user experience (Ogawa et al., 2009). The present paper is a revised and expanded version of Ogawa et al. (2009); it provides three contributions. First, on the basis of actual data traffic analysis, we model download traffic of thin-client systems by using a two-state model with interactive data flows in response to keystrokes and bulk data flows related to screen updates. Since users are more sensitive to the keystroke response time, our objective is to minimise the latency of interactive data flows without increasing that of bulk data flows. Second, through simulation experiments using real field data, we investigate the primary delays during transfers of interactive data flows over a WAN. We then demonstrate that the primary delays are queuing delay in the router and TCP buffering delay in the server, which is caused by interactions between interactive and bulk data flows. Third, we describe comprehensive approaches for performance improvement. We first evaluate the effectiveness of existing options: priority queuing of interactive data flows and use of the TCP Selective Acknowledgment (SACK) option. Although these techniques reduce the probability of short delays occurring, a packet of an interactive data flow is sometimes held in the server for more than a second. We then propose two TCP mechanisms to reduce lengthy delays without modifying the remote desktop protocol itself: modifying the retransmission timeout calculation and enhancing the SACK control.

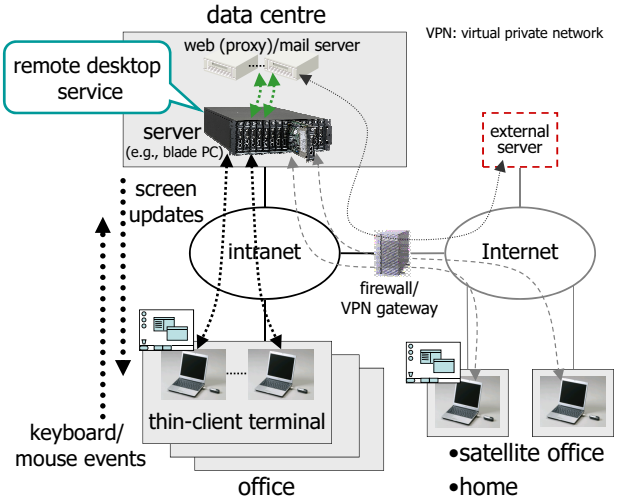


Figure 1 Overview of thin-client system

The rest of this paper is organised as follows. We introduce our traffic model in Sec. 2. Then we evaluate the end-to-end delays without modifications in Sec. 3, with the existing options in Sec. 4, and with the addition of our improved TCP mechanisms in Sec. 5. We also evaluate the effect of WAN propagation delay in Sec. 6. We conclude and discuss remaining issues in Sec. 7.

2 Modelling of Thin-Client Traffic

To improve the performance related to keystrokes, we begin by identifying the data flows corresponding to

them and by defining a metric of the flow performance. In this section, we therefore introduce a traffic model for the thin-client data flows and configure the input data for performing the simulation experiments described in following sections. We also define the usability metric for evaluating the experimental results.

2.1 Two-State Modelling

This section is based on a one-month observation of our test system (Figure 1). The system was composed of about 200 thin-client/server pairs and their communication traffic aggregated at the core switch of the data centre was monitored using a network protocol analyser (Wireshark®, 2011). Each server, implemented by a blade PC, ran Microsoft® Windows® XP. Each client had access to the corresponding server via Microsoft® RDP. Most of the applications executed in the servers were typical desktop applications such as email clients, web browsers, and Microsoft® Office.

We constructed the model by targeting only response packets from the servers, not request packets to them, because the traffic due to request packets is much lower than the response packet traffic. Network congestion therefore arises when there are many response packets. Although the request packets may be delayed because of background traffic, this consideration is outside the scope of this paper. Furthermore, to investigate the effect of network latency, we started with a model of response packets not affected by network latency, i.e., the delay of the TCP ACK response, but sent directly corresponding to the user's usual keyboard/mouse input rate (as if users were using traditional fat clients). We therefore analysed only packets traversing the intranet, where almost all connections had a round-trip time (RTT) in milliseconds and an end-to-end bandwidth of nearly 100 Mbps.

Observation of data traffic revealed that response packet transfers could be classified roughly into two types, as shown in Figure 2. Interactive data transfer is used mainly for character information delivery. An interactive data flow is defined here to carry a single response packet of character information in response to a keystroke (like Telnet traffic). Meanwhile, bulk data transfer is used mainly for screen update information delivery. A bulk data flow is defined to be composed of a block of less than about 100 response packets representing screen updates (like HTTP (web) traffic). In our test system, we found that whatever desktop application programs the server was executing, data flows of response packets could be expressed as a mixture of these two types of data flows, although the frequency of each flow depended on the application program used. We therefore considered that the data flows could be modelled using a two-state system, as shown in Figure 3, where α is the state transition probability of changing from sending an interactive data flow to sending a bulk data flow and β is that of the opposite change. In addition, p_I is the probability of sending an interactive data flow and p_B is that of sending a bulk data flow.

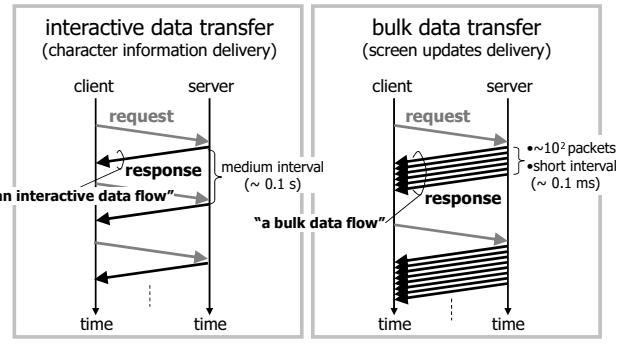


Figure 2 Two types of data transfer in thin-client system

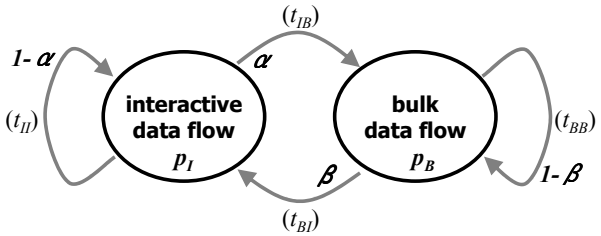


Figure 3 Two-state model of data flows

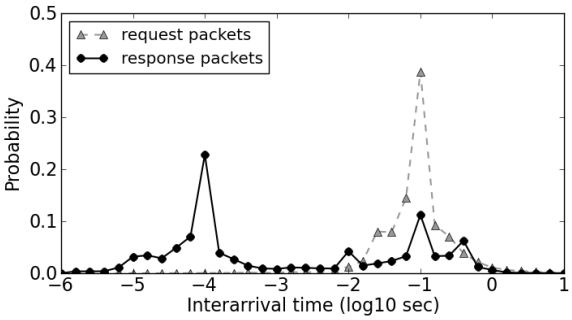


Figure 4 Interarrival time distributions of request and response packets

Moreover, t_{II}, t_{IB}, t_{BI}, and t_{BB} are respectively the intervals between sending an interactive data flow and the next flow, between sending an interactive data flow and a successive bulk data flow, between sending a bulk data flow and a successive interactive data flow, and between sending a bulk data flow and the next flow.

To identify the two types of data flows, we analysed the interarrival time distributions for response packets as well as request packets, as shown in Figure 4. The maximum time interval between two adjacent packets was set to 60 s. Furthermore, TCP acknowledgment (ACK) packets were excluded. The response packet distribution has roughly two peaks. The peak centred around 10^{-1} s overlaps with that of the request packet distribution, which should be related to the average frequency of up-and-down keystrokes. Meanwhile, the peak at 10^{-4} s is related to the interarrival time of two

Table 1 Transition probability and interval of response traffic (average for 100-sample data, each 300 s long)

current data flow	next data flow	
	interactive	bulk
interactive	0.91	0.09 (α)
	0.27 s (t_{II})	0.22 s (t_{IB})
bulk	0.42 (β)	0.58
	0.24 s (t_{BI})	0.11 s (t_{BB}) *

* Average interval between two packets in a block was 0.36 ms.

Table 2 Features of response traffic (average for 100-sample data, each 300 s long)

	interactive	bulk
no. of flows	934.1 (82% (p_I))	198.4 (18% (p_B))
no. of pkts	934.1 (39%)	1,467.0 (61%)
	$\langle$1.0/flow$\rangle$	$\langle$7.4/flow (c_B)$\rangle$
bytes	128,189 (8.5%)	1,373,366 (91.5%)
	$\langle$137.2/flow$\rangle$	$\langle$6922.2/flow$\rangle$
	$\langle\langle$137.2/pkt$\rangle\rangle$	$\langle\langle$936.2/pkt$\rangle\rangle$

consecutive packets in a block of response packets, i.e., a bulk data flow. We could therefore identify interactive and bulk data flows by setting a threshold for the interarrival time of response packets. If the interarrival time was longer than the threshold, the packet was judged to be an interactive data flow or the head of a bulk data flow. We set the threshold to $10^{-2.2}$ s (6.3 ms) on the basis of the correspondence between the request and response packet distributions. By using the threshold, we classified one-month observed response packets into the two data flow types; this revealed that the average state transition probabilities were $\alpha = 0.09$ and $\beta = 0.42$.

To perform simulation experiments, we extracted the top hundred 300-s-long series of response packets whose state transition probabilities (α and β) were closest to the overall average probabilities. The average features of the extracted data are shown in Tables 1 and 2. An overview of the traffic feature is that interactive data flows predominated in terms of the number of flows while bulk data flows predominated in terms of bytes. The parameters in the tables, t_{II}, t_{IB}, t_{BI}, t_{BB}, p_I, p_B, and c_B (where c_B is the average number of packets included in a bulk data flow), which were calculated from the extracted data, were used to calculate Eq. (4) in Sec. 2.2.

2.2 Usability Metric

According to Tolia et al. (2006), thin-client users notice the response time when it exceeds 150 ms, which corresponds to the *human response time* (McCabe, 2003); furthermore, they become frustrated and less productive when it exceeds 1 s. We therefore set two thresholds for packet delay in an interactive data flow:

$$\text{Unnoticeable}: \quad < (150 - t_0) \text{ ms} \qquad (1)$$

$$\text{Productivity degrading}: \quad > (1000 - t_0) \text{ ms}, \qquad (2)$$

where t_0 is the time for transferring a request packet from a client to a server. With these thresholds, two usability metrics for evaluating the performance of interactive data flows are defined as the ratio of the number of packets whose delays are longer than each threshold to the total number of packets in all the interactive data flows tested; these ratios are called $r_{I_{150}}$ for Threshold (1) and $r_{I_{1000}}$ for Threshold (2).

Furthermore, by using the two-state model, we can convert each metric to the average cycle of time during which the response time threshold is exceeded in order to understand the evaluation results intuitively. After a long enough period has elapsed, $p_I = \beta/(\alpha + \beta)$ and $p_B = \alpha/(\alpha + \beta)$, respectively. The relationship between the number of interactive data flows, f_I, which includes n_I packets, and the number of bulk data flows, f_B, which includes n_B packets, becomes $f_I : f_B \simeq p_I : p_B$, $f_I = n_I$, and $f_B = n_B/c_B$. The average cycle of time T for completely sending f_I interactive data flows and f_B bulk data flows in the steady state is

$$T = (1-\alpha) f_I t_{II} + \alpha f_I t_{IB} + \beta f_B t_{BI} + (1-\beta) f_B t_{BB}. \qquad (3)$$

Consequently, the average cycle of time exceeding the threshold, $T_{I_{150}}$ or $T_{I_{1000}}$, can be expressed as a function of the $r_{I_{150}}$ or $r_{I_{1000}}$, respectively, as follows.

$$T_{I_k} = \frac{(1-\alpha)t_{II}}{r_{I_k}} + \frac{\alpha t_{IB}}{r_{I_k}} + \frac{\beta p_B t_{BI}}{c_B p_I r_{I_k}} + \frac{(1-\beta) p_B t_{BB}}{c_B p_I r_{I_k}} \qquad (4)$$
$$(I_k = I_{150} \text{ or } I_{1000})$$

3 End-to-End Delay Analysis

Using the traffic model in Sec. 2, we evaluated the delay of the current system through simulation experiments using ns-2 (release 2.33) (ns-2, 2011).

3.1 Network and System Model in Experiments

Our target system has a typical architecture for a Japan-wide intra-firm system (Figure 1). It is reasonable to assume that network traffic over the wide-area intranet comprises only traffic between the servers and thin clients since the communication area for the traffic to and from other servers, such as web/mail servers, is mostly limited to the data centre and other logical links. Given these considerations, we chose to use the simulation model shown in Figure 5. This model consists of 100 pairs of a server (sender) and a thin client (receiver) plus two routers and links between them. The networks in the data centre and in the remote office had a link bandwidth of 100 Mbps and a propagation delay of 1 ms. The bandwidth of the link between the two routers, i.e., the bottleneck link corresponding to the wide-area intranet such as one between Tokyo and Osaka, was set at 10 Mbps, and its propagation delay was 20 ms. An ns-2 *FullTCP* agent running on each server simultaneously transmitted a 300-s-long series of response packets

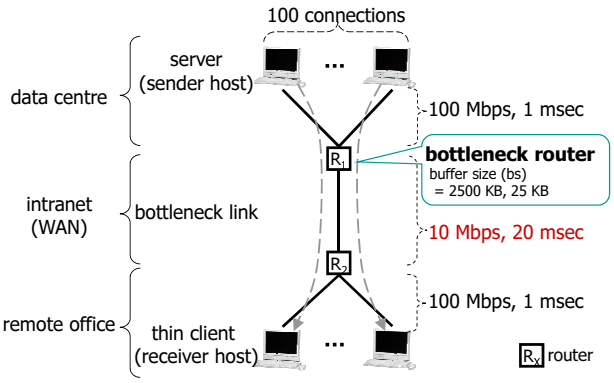


Figure 5 Network and system model in simulation experiments

extracted in Sec. 2.1 to a corresponding agent running on each client over a TCP Reno connection. A tail-drop (FIFO) discipline was used at the router's buffer. The buffer size was set to two different values: the bandwidth-delay product, i.e., 25 Kbytes, or a sufficiently large value of 2500 Kbytes. We repeated the 300-s simulation seven times with the transmission start time of each sender agent shifted at 0.25-ms intervals because we expected the simulation result to change depending on the degree of synchronisation between packets from different senders in the router buffer.

Note that the *FullTCP* agent was used because it counts a sequence in bytes rather than in packets. For convenience in our experiments, the agent on the server labelled packets as "interactive" or "bulk" on the basis of the sending interval. We shall abbreviate the buffer size as "bs" in the figures. We shall also call the router connected to the bottleneck link the "bottleneck router" or simply the "router" here. In addition, to make the model simple, we set the TCP settings so as to turn off the Nagle algorithm, delayed acknowledgments, and slow-start restart. Furthermore, additional experiments with several WAN propagation delays other than 20 ms were executed, as mentioned in Sec. 6.

3.2 Factors of End-to-End Delay

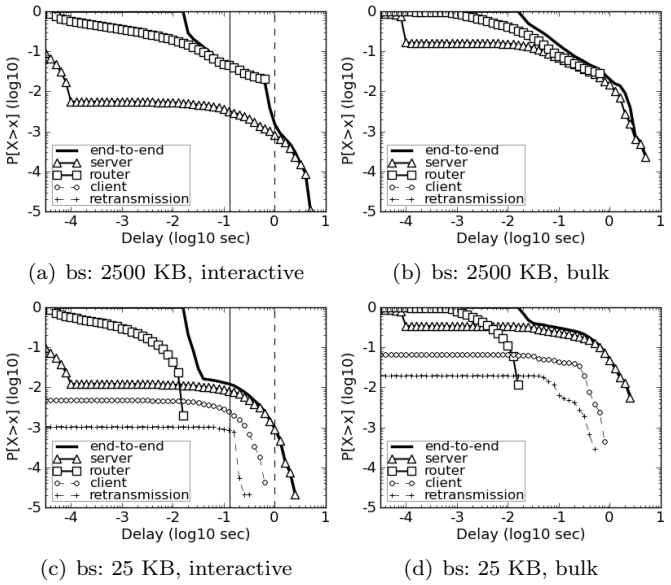
Let us enumerate the factors contributing to the end-to-end delay of a response packet. The end-to-end delay is defined here as the one-way trip time from when an application program on the server sends a response packet to when the application program on a client receives it. This delay consists of delays related to the server, router, client, retransmission, and propagation in each link. The delay in the server is considered to be the sum of the packet buffering and transmission delays in the server because the processing delay is relatively small in our model. The delay in the router is similarly defined as the sum of the packet buffering (queuing) and transmission delays in the router. The delay in the client is buffering delay, which is the time during which the packet is buffered in the TCP layer before being sent to the application layer. We also analysed the delay for packet retransmission initiated by TCP, which is the time from when a packet is dropped at the router to when a copy of the packet reaches the router. The delay in the client is called "head-of-line blocking" and does not occur unless the packet receiving order is switched owing to subsequent packets leaving earlier the router. This can be simply regarded as the movement of the delay source from the router to the client and does not increase the end-to-end delay. Therefore, we did not try to reduce it in this study.

3.3 Delay Distribution without Modifications

One set of the typical simulation results is shown in Figure 6. It is plotted in the form of complementary cumulative distribution functions, i.e., ratio of the count of packets whose delays are larger than the value indicated on the horizontal axis to the total packet count. Graphs (a) and (b) show the distributions for the end-to-end delay, the delay in the server, the delay in the (bottleneck) router, the delay in the client, and the retransmission delay for a buffer size of 2500 Kbytes (large buffer case). Graphs (c) and (d) are for a buffer size of 25 Kbytes (small buffer case). Furthermore, graphs (a) and (c) are for interactive data flows, while graphs (b) and (d) are for bulk data flows. In graphs (a) and (c), Thresholds (1) and (2) for interactive data flows, i.e., 130 ms and 980 ms (which have had the time for transferring a request packet across WAN subtracted from them), are specified. Moreover, two ratios $r_{I_{150}}$ (the ratio of end-to-end delays exceeding 130 ms) and $r_{I_{1000}}$ (the ratio of ones exceeding 980 ms) are presented below in the from of average values for seven experiments.

Large Buffer Case. The changes in queue length at the router indicate that bursty traffic continuing for several hundred milliseconds (up to about 1 s) occurred periodically. This change rate was rapid for packets sent at average intervals of about 300 ms (see Table 1). The maximum queue length during the 300-s simulation period was about 400 Kbytes; this means that the maximum queuing delay was around 300 ms. The average utilisation of the bottleneck link over the entire simulation period was 54%, while that for interactive data flows was 3.5%.

For interactive data flows, $r_{I_{150}}$ was 3.6×10^{-2} and $r_{I_{1000}}$ was 1.7×10^{-3}. As shown in Figure 6(a), at the threshold of 130 ms, the end-to-end delay was mainly the delay in the router. Meanwhile, around 980 ms, the delay in the server was predominant. As shown in Figure 6(b), the delay in the server was larger for bulk data flows than for interactive data flows. The main component of the end-to-end delays exceeding 100 ms was the delay in the server. This delay was caused by the input of large blocks of packets into the TCP buffer; this made the buffering delay so large that the following bulk data flows as well as interactive data flows experienced possible delays. This type of delay seems reasonable because users



(a) bs: 2500 KB, interactive (b) bs: 2500 KB, bulk

(c) bs: 25 KB, interactive (d) bs: 25 KB, bulk

Figure 6 Delay distributions without modifications

sometimes experience non-smooth text display on thin-client terminals in a high-network-latency environment when they simultaneously start application programs for showing moving pictures and cause the screens to be updated frequently. Moreover, the delay in the server was even longer because TCP retransmission timeouts occurred without packet being dropped. Details are given in the next section.

Small Buffer Case. For interactive data flows, $r_{I_{150}}$ was 1.2×10^{-2} and $r_{I_{1000}}$ was 9.0×10^{-4}. Because a smaller buffer in the router caused less delay but more drops, the buffering delay in the router was smaller and that in the server was larger compared with the large buffer case, as shown in Figures 6(c) and 6(d). When a server transmitted large bulk data flows composed of more than a few dozen packets or more than a few hundred packets in some cases, many packets were dropped in less than a second. Furthermore, such bursty drops prevented packets sent by other servers from entering the router buffer. As a result, the server transmitting the large bulk data flows shrank its TCP *cwnd* repeatedly, and other servers shrank theirs as well. This increased the buffering delay in the servers of the bulk data flows and subsequent interactive data flows.

4 Existing Options and Problems

In this section, we present appropriate options developed from the existing techniques and evaluate them in the same manner as in the previous section.

4.1 Existing Techniques

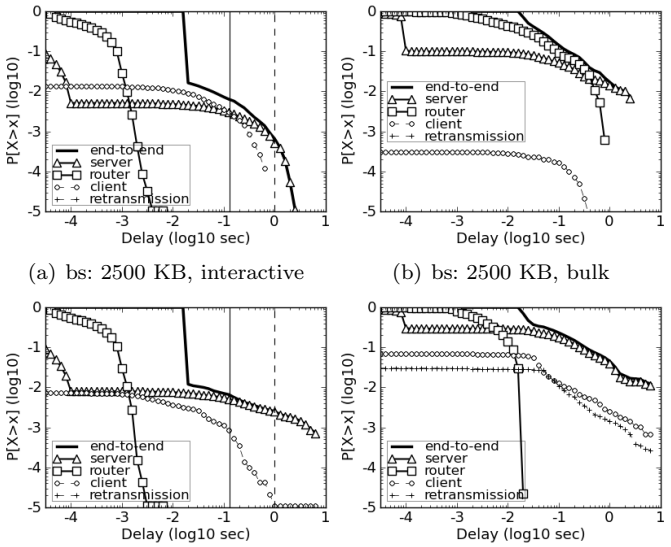
Prioritising Interactive Data Flows. The delay in the router described in large buffer case in Sec. 3.3 can be reduced by implementing priority queuing in the router. This means using two buffers that share the available buffer size: a higher-priority buffer for processing interactive data flows and a lower-priority buffer for processing bulk data flows. Packets in the higher-priority buffer are always processed ahead of packets in the lower-priority buffer. Accordingly, packets of interactive data flows are forwarded with a minimal delay at the router. The holding time of packets in bulk data flows in the lower-priority buffer increases by only a few milliseconds because of the small volume of interactive data flows. Moreover, when the router does not have a large buffer, an interactive data flow can be sent without any possibility of its packets being dropped.

TCP SACK Option. As explained in the small buffer case in Sec. 3.3, if the router does not have a large buffer, multiple packets of a large bulk data flow may be dropped at the router, which in turn causes a delay in the server. This delay can be reduced by applying the TCP SACK option to the server so that it can recover dropped packets quickly and keep *cwnd* large (Mathis et al., 1996).

4.2 Evaluation Results

Large Buffer Case. For interactive data flows, $r_{I_{150}}$ decreased to 6.1×10^{-3} because the buffering delay in the router was reduced by priority processing. In contrast, $r_{I_{1000}}$ was 5.5×10^{-4}, where the average value was slightly improved but some values became much worse. This sort of delay was mainly buffering delay in the server, as shown in Figure 7(a), and it was further increased by retransmission timeouts in spite of no packets being dropped. Figure 8 shows a server's behaviour when it invoked such a retransmission timeout. The upper graph shows changes in the sequence number of packets sent by the server and changes in the server's *cwnd*. The lower graph shows changes in the parameters used for calculating the retransmission timeout value *RTO*, i.e., *rtt*, *srtt*, *rtttvar* (see Eqs. (5)–(7) in Sec. 5.1), and the changes in the router's queue length. This sort of timeout occurred (***A*** in Figure 8), because *srtt* and *rttvar*, and hence *RTO*, did not quickly become sufficiently large when large bulk data flows entered the router's buffer and the router's queue length increased (***B*** in Figure 8). Moreover, the RTT changed abruptly as a result of interactive data flow prioritisation when bulk data flows were switched to interactive ones (***C*** in Figure 8). These timeouts were concentrated on several servers sending bulk data flows at that time because the bulk data flows were always placed at the end of the router's queue. Such timeouts sometimes affected a server more than five times within a few seconds.

Small Buffer Case. For interactive data flows, $r_{I_{150}}$ decreased to 6.4×10^{-3} because the delay in the servers was reduced by using the TCP SACK option. In contrast,



(a) bs: 2500 KB, interactive (b) bs: 2500 KB, bulk

(c) bs: 25 KB, interactive (d) bs: 25 KB, bulk

Figure 7 Delay distributions with prioritised interactive data flows and TCP SACK option

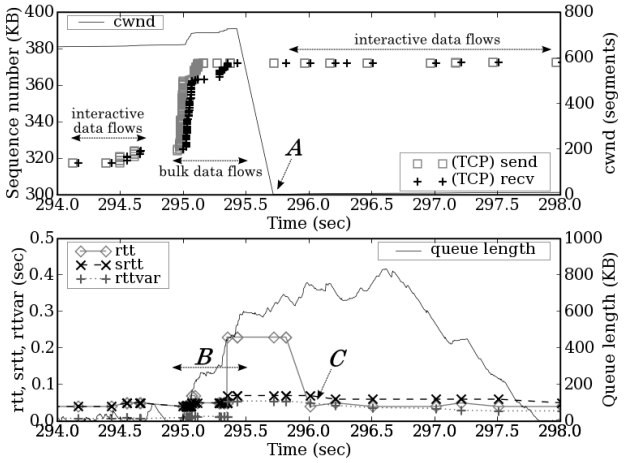


Figure 8 Retransmission timeout occurred without packet drops: changes in the sequence number of packets sent by a server and in the server's *cwnd* (upper) and changes in the server's *RTO* parameters and in router queue length (lower)

$r_{I_{1000}}$ increased to 2.2×10^{-3} as a result of the significant delays for buffering the preceding bulk data flows at the server, as shown in Figure 7(c). These buffering delays occurred because the servers sending large bulk data flows experienced bursty drops (hundreds of drops in some cases) at the router. These packet drops were concentrated at a few servers sending bulk data flows at that time.

The server using the TCP SACK option quickly retransmitted such dropped packets. When retransmitted packets were also dropped, the retransmission timeout triggered their retransmissions because the TCP fast retransmission mechanism is

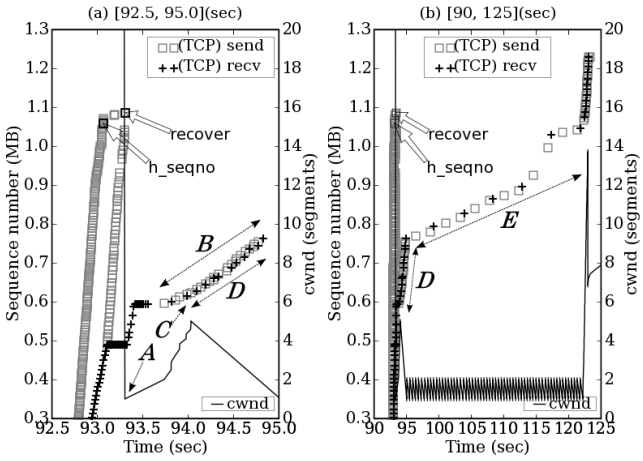


Figure 9 Server buffering delays of several seconds caused by bursty packet drops: magnified view of changes in the sequence number of packets sent and received by a server and changes in the server's *cwnd* (left) and their overall view (right)

(Packets sent and received are plotted once every 10 packets)

not applied to lost retransmissions. Figure 9 illustrates two problems causing several-second delays after the retransmission timeout. The left graph shows changes in the sequence number of packets sent and received by the server and changes in the server's *cwnd* from 92.5 s to 95.0 s, which indicates the retransmission timeout. The right graph shows those from 90 s to 125 s; this is an overview of the several-second delays in the server. In the graphs, the packets are plotted once every ten packets for clarity. These problems were due to the TCP SACK implementation (ns-2's *FullTCP* is similar to 4.x BSD TCP (ns-2, 2011)). The first problem occurred when the server received three duplicate ACKs and entered fast recovery mode (***C*** in Figure 9). The server did not increase *cwnd* when it received an ACK whose value was lower than *recover* in fast recovery mode (***D*** in Figure 9), where *recover* was the highest sequence number transmitted by the server when the first timeout occurred (***A*** in Figure 9). The second problem happened after that. When the server received an ACK whose value was lower than *recover*, it determined which packet to retransmit by comparing *h_seqno* with the SACK blocks reported by the paired client, where *h_seqno* is the highest sequence number indicating a hole that has not yet been filled by the fast retransmissions when the first timeout occurred. The server sent the packet pointed to by the ACK when the *h_seqno* was higher than all SACK blocks (***B*** in Figure 9). Furthermore, the server would have sent the packet corresponding to the *h_seqno* when the *h_seqno* was lower than a hole reported by the SACK blocks, but it could not do so because *cwnd* was not large enough to accommodate *h_seqno* at that time. As a result, the server waited for extra timeouts (***E*** in Figure 9).

5 Proposed Mechanisms and Evaluation

The problems described in the previous section are due to the traffic balance between interactive and bulk data flows shown in Tables 1 and 2. We thus developed several mechanisms and evaluated them in the same way.

5.1 Mechanisms for Reducing Delay in Server

Modifying the Retransmission Timeout Value Calculation. The buffering delay in the server described in the large buffer case in Sec. 4.2 can be decreased by keeping the server's *cwnd* large enough. This can be done by preventing the retransmission timer from expiring. The timer expires when *RTO* does not follow a rapid increase in the delay of the router's buffer owing to bulk data flows. Here, *RTO* is calculated on the basis of the measured RTT for the given connection as follows (Stevens and Wright, 1995).

$$srtt \leftarrow (1-g)srtt + g \cdot rtt \tag{5}$$

$$rttvar \leftarrow (1-h)rttvar + h|rtt - srtt| \tag{6}$$

$$RTO \leftarrow srttt + 4 \cdot rttvar, \tag{7}$$

where *rtt* is the latest measured RTT, *srtt* is the current smoothed RTT estimator, and *rttvar* is the smoothed mean deviation estimator. The gains g and h are usually set to 1/8 and 1/4, respectively. In the proposed mechanism, *srtt* is updated by using g:

$$g = \begin{cases} 7/8 & (rtt \geq srtt) \\ 1/8 & (rtt < srtt). \end{cases} \tag{8}$$

As a result, *RTO* reflects the latest RTT, and it keeps pace with any rapid increase in the router queue when large bulk data flows are input. Moreover, *RTO* does not become too small when data flows switch from bulk to interactive under the interactive data flow prioritisation setting.

Furthermore, the server initialises *RTO* after the idle period in order to match the change in queue length after a long interval between two data flows. We use the time when the server invokes TCP's slow-start restart to determine the idle period. At that time, the server initialises *srtt* and *rttvar*, but not *cwnd*.

Temporarily Turning Off TCP SACK Control. Significant buffering delays in the server described in small buffer case in Sec. 4.2 can be avoided by having the server recover from bursty drops as quickly as possible. This can be achieved by steadily increasing *cwnd* and not invoking any extra timeouts. Some studies have developed SACK mechanisms to detect and recover a lost retransmission (Lin and Kung, 1998; Kim et al., 2005). However, they are not appropriate for recovering bursty packet drops in the present case because they consider random multiple drops. We therefore temporarily turn off the TCP SACK control after a timeout occurs and do not turn it back on until the dropped packets have been recovered, as follows.

- To enable the server to increment *cwnd* when it receives an ACK whose value is lower than *recover* after the timeout, the server is configured to not enter fast recovery mode if it receives three duplicate ACKs after the timeout.
- To prevent the server from becoming incapable of sending a packet after the timeout, the server is configured to send the packet pointed to by an ACK, rather than selecting one by comparing the *h_seqno* and the SACK blocks, if the ACK value is lower than *recover* after the timeout.

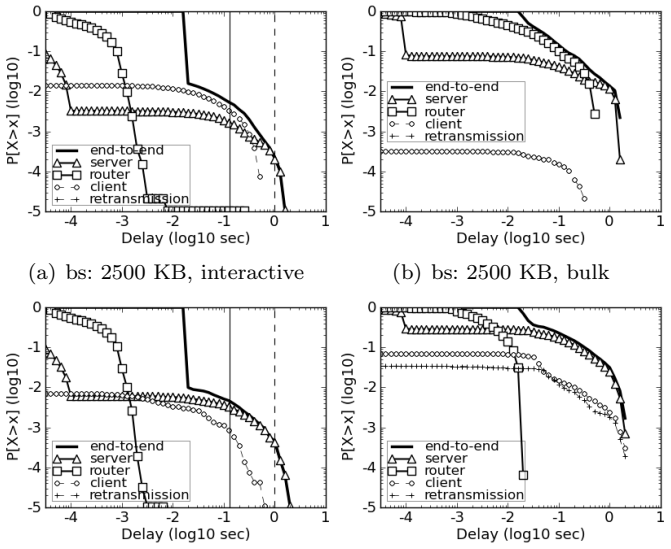
Since these modified processes are similar to those of TCP Reno, traffic burstiness during the modified period is considered to be equivalent to that of TCP Reno. Furthermore, an increase in burstiness is related to bulk data flows. The bulk data flows are given lower priority and thus do not affect the simultaneous interactive data flows at the router buffer.

5.2 Evaluation of Proposed Mechanisms

The underlying cause of the server delay described in Sec. 4.2 is that interactive and bulk data flows coexist in a TCP connection. We therefore additionally propose separating the TCP connection and evaluate the proposed mechanisms for two cases. Case 1: a single TCP connection is shared between interactive and bulk data flows; this is the current specification. Case 2: the TCP connection for interactive data flows is separated from that for bulk; this means the application protocols need to be modified. Note that Case 2 is impractical when the thin-client system is based on a proprietary application protocol (as in the case of this paper), whereas Case 1 with the proposed mechanisms can be applied by using a transport-layer proxy mechanism (Yamanegi et al., 2005). The results for Case 2 are therefore used for clarifying the limitations of Case 1.

5.2.1 Case 1: Shared TCP Connection

Large Buffer Case. For interactive data flows, $r_{I_{150}}$ was 5.0×10^{-3}, which means that this ratio was slightly improved by applying only priority queuing. Moreover, $r_{I_{1000}}$ decreased to 2.1×10^{-4}, where the negative effect of the existing option disappeared. Figure 11 depicts an example of avoiding TCP timeouts without packet drops, which was obtained under the same experimental conditions as in Figure 8. As shown in the lower graph, *srtt* quickly coped with its rapid increase for bulk data flows in the router buffer (***A*** in Figure 11). Subsequently, it decreased slowly even though *rtt* suddenly dropped owing to the switch to interactive data flows (***B*** in Figure 11). These mechanisms prevented the server's retransmission timer from expiring and stopped its *cwnd* from shrinking. They thus reduced the number of timeouts experienced by all servers. The average number of timeouts over the 300-s simulation period was 3.1 per connection when we used priority queuing for the



(a) bs: 2500 KB, interactive (b) bs: 2500 KB, bulk

(c) bs: 25 KB, interactive (d) bs: 25 KB, bulk

Figure 10 Delay distributions with prioritised interactive data flows, TCP SACK option, and proposed mechanisms for shared TCP connections

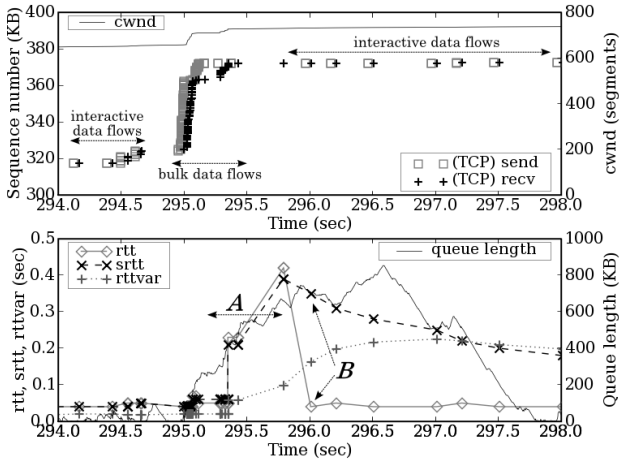


Figure 11 Avoidance of retransmission timeout shown in Figure 8: changes in sequence number of packets sent by a server and in the server's *cwnd* (upper) and changes in the server's *RTO* parameters and in router's queue length (lower)

interactive data flows and the TCP SACK option, and this value decreased to 1.9 per connection when the proposed mechanisms were used as well. As a result, the ratio of end-to-end delays exceeding 980 ms for bulk data flows in the servers fell to the level before the priority queuing was configured in the router, as shown in Figure 10(b), which decreased $r_{I_{1000}}$ as shown in Figure 10(a).

Small Buffer Case. For interactive data flows, $r_{I_{150}}$ slightly decreased to 4.6×10^{-3} from the value for the case where only the TCP SACK option was applied. Furthermore, $r_{I_{1000}}$ greatly decreased to 4.0×10^{-4}. The left graph of Figure 12 shows a faster recovery from

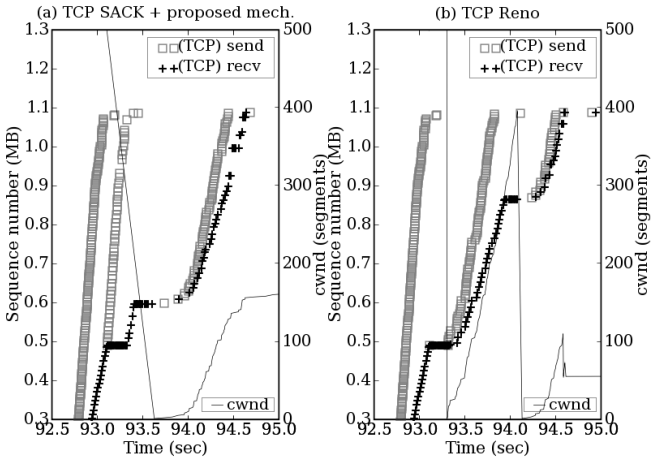


Figure 12 Faster recovery from bursty packet drops shown in Figure 9 by using the TCP SACK option and proposed mechanisms (left) and by using TCP Reno (right)

(Packets sent and received are plotted once every 10 packets)

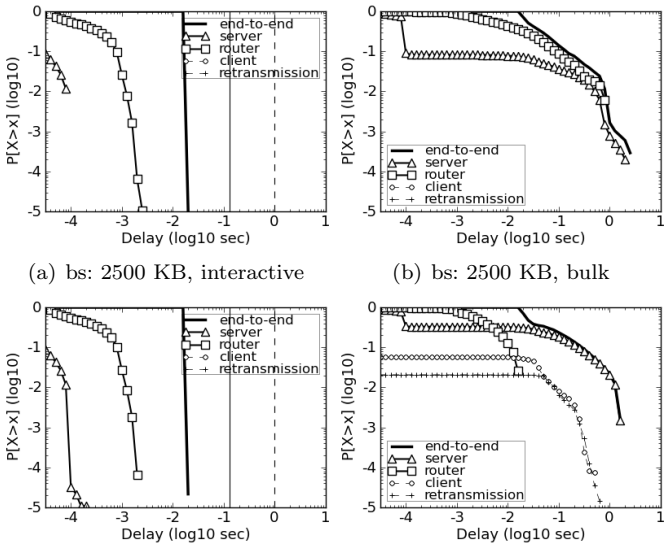
bursty packet drops, under the same setting as in Figure 9. The recovery speed, i.e., burstiness, with this sequence was nearly as high as when using TCP Reno in the server, as shown in the right graph of Figure 12. As a result, the several-second delays in the servers for transmitting bulk data flows were eliminated, as shown in Figure 10(d). This reduced the over-980-ms delays in the server and thus the end-to-end delays for interactive data flows, as shown in Figure 10(c).

5.2.2 Case 2: Separate TCP Connections

We assumed that the client application did not have to maintain the packet order between the interactive and bulk data flows sent by the server application. In both buffer cases, the end-to-end delays for interactive data flows were almost completely due to propagation delay; these delays were much lower than 130 ms, as shown in Figure 13. Since interactive data flows were transferred using a dedicated TCP connection, the delay for interactive data flows was independent of that for bulk data flows.

6 Effect of WAN Propagation Delay

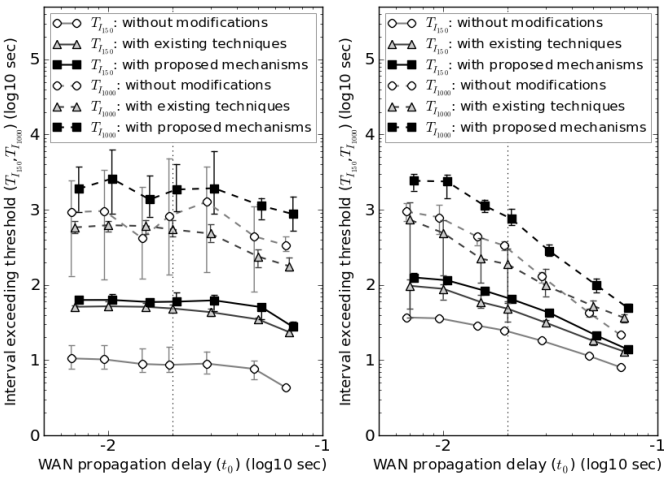
Finally, we evaluate the effect of WAN propagation delay on the results for a single TCP connection shared by interactive and bulk data flows without modifications (Sec. 3.3) with the existing techniques (Sec. 4.2) and with the proposed mechanisms (Sec. 5.2.1). Here, we clarify the improvements among these three approaches. For each case, we performed additional experiments with several WAN propagation delays ranging from 7 to 70 ms. The evaluation results were converted into the average intervals between delays exceeding either Threshold (1) or (2), i.e., $T_{I_{150}}$ and $T_{I_{1000}}$ by using Eq.



(a) bs: 2500 KB, interactive (b) bs: 2500 KB, bulk

(c) bs: 25 KB, interactive (d) bs: 25 KB, bulk

Figure 13 Delay distributions with prioritised interactive data flows, TCP SACK option, and proposed mechanism for separate TCP connections



(a) bs: 2500 KB, interactive (b) bs: 25 KB, interactive

Figure 14 Interval between delays exceeding threshold ($T_{I_{150}}$: solid lines and $T_{I_{1000}}$: dashed lines (Eq. (4)))

(Horizontal positions are adjusted for error bars visibility.)

(4). This means that the longer the interval becomes the less often users notice the delay and get frustrated. Note that the probability of delays exceeding the threshold, and hence the intervals as well, is shared by all users in the system, rather than entirely affecting each user.

Figure 14 shows changes in the intervals for interactive data flows, where each marker specifies the mean and each error-bar specifies the maximum and minimum values for seven experimental runs for each setting. When the WAN propagation delay increased, its effect was not so obvious in the large buffer case; meanwhile, the intervals became shorter along with the delay in the small buffer case. In the large and small buffer cases, average $T_{I_{150}}$ values with the proposed mechanism were 55 and 67 s; these were 6.5 and 2.8 times longer, respectively, than those for without modification. Moreover, they both slightly increased the improvement obtained by applying only the existing techniques. Furthermore, average $T_{I_{1000}}$ values with the proposed mechanism were 1.7×10^3 and 1.0×10^3 s; these were 3.6 and 3.9 times longer, respectively, than those for with only the existing techniques. In addition, the variation in $T_{I_{1000}}$ without modification in the large buffer case and that with the existing techniques at less than 20 ms in the small buffer case were both considerably large if large bulk data flows from some servers reached the router in a highly synchronised manner for a few milliseconds. Meanwhile, with the proposed mechanisms, the variation was relatively small. These results show that the proposed mechanisms negate the shortcomings of the existing approaches and improve upon them. In addition, for bulk data flows, the intervals obtained with the proposed mechanisms were equivalent to other results.

7 Conclusion

The thin-client system has a particular traffic pattern involving a mixture of interactive and bulk data flows. This study aimed to improve the performance of interactive data flows while not influencing bulk data flows. We found that the average end-to-end delay of the interactive data flows could be reduced by applying priority queuing and utilising the TCP SACK option. The occurrences of lengthy delays exceeding about 1000 ms resulting from the buffering delay in the server could be further reduced by modifying the retransmission timeout calculation and temporarily halting the TCP SACK control. Nevertheless, to completely eliminate the effect of bulk data flows on interactive data flows, it is necessary to establish separate TCP connections for these two types of data flows. The remote desktop protocol can change its setting and reduce the traffic volume of bulk data flows to cope with a low-bandwidth environment. Researchers have previously attempted to improve performance by caching screen updates (Yang and Tiow, 2007; Vankeirsbilck et al., 2008). However, these measures do not reduce the bulk data flow volume to the level of the interactive data flow volume. Thus, delays caused by interactions between interactive and bulk data flows will still occur as long as the two flow types share a TCP connection. In the future, we would like to investigate network architectures for *cloud computing* using thin-client technology.

Ltd. Wireshark is a registered trademark of the Wireshark Foundation.

References

Braden, R. (1989). Requirements for Internet Hosts - Communication Layers, RFC 1122.

Heidemann, J. (1997). Performance interactions between P-HTTP and TCP implementations, *ACM SIGCOMM Computer Communication Review* **27**(2): 65–73.

Jacobson, V. and Karels, M. J. (1988). Congestion avoidance and control, *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, ACM, pp. 314–329. slightly-revised 1992 version of the 1988 paper.

Kim, B., Kim, Y., Oh, M. and Choi, J. (2005). Microscopic behaviors of TCP loss recovery using lost retransmission detection, *Consumer Communications and Networking Conference (CCNC)*, pp. 296–301.

Lai, A. M. and Nieh, J. (2006). On the performance of wide-area thin-client computing, *ACM Transactions on Computer Systems* **24**(2): 175–209.

Lin, D. and Kung, H. (1998). TCP fast recovery strategies: Analysis and improvements, *Proc. of INFOCOM*, Vol. 1, pp. 263–271.

Mathis, M., Mahdavi, J., Floyd, S. and Romanow, A. (1996). TCP Selective Acknowledgment Options, RFC 2018.

McCabe, J. D. (2003). *Network Analysis, Architecture and Design, Second Edition*, San Francisco:Morgan Kaufmann Publishers Inc.

Minshall, G., Saito, Y., Mogul, J. C. and Verghese, B. (1999). Application performance pitfalls and TCP's Nagle algorithm, *Proc. of 2nd Workshop on Internet Server Performance*, pp. 36–44.

Mogul, J. C. and Minshall, G. (2001). Rethinking the TCP Nagle algorithm, *ACM SIGCOMM Computer Communication Review* **31**(1): 6–20.

MSDN® (2011). Remote Desktop Protocol, `http://msdn.microsoft.com/en-us/library/aa383015(VS.85).aspx`. [accessed 21 March 2011].

Nagle, J. (1984). Congestion control in IP/TCP internetworks, RFC 896.

Nieh, J., Yang, S. J. and Novik, N. (2003). Measuring thin-client performance using slow-motion benchmarking, *ACM Transactions on Computer Systems* **21**(1): 87–115.

ns-2 (2011). The Network Simulator - ns-2, `http://nsnam.isi.edu/nsnam/index.php/Main_Page`. [accessed 21 March 2011].

Ogawa, Y., Hasegawa, G. and Murata, M. (2007). Transport-layer optimization for thin-client systems, *Proc. of IEEE CQR 2007 Workshop*.

Ogawa, Y., Hasegawa, G. and Murata, M. (2009). A transport layer approach for improving interactive user experience on thin clients, *Proc. of Australasian Telecommunication Networks and Applications Conference (ATNAC 2009)*, Canberra.

Richardson, T., Stafford-Fraser, Q., Wood, K. R. and Hopper, A. (1998). Virtual Network Computing, *IEEE Internet Computing* **2**(1): 33–38.

Sagawa, N. and Koda, K. (2009). Toward human-oriented office, *Hitachi Review* **58**(4): 169–173.

Scheifler, R. W. and Gettys, J. (1997). *X Window System: Core and Extension Protocols : X Version 11, Releases 6 and 6.1*, Oxford:Butterworth-Heinemann.

Schlosser, D., Binzenhofer, A. and Staehle, B. (2007). Performance comparison of windows-based thin-client architectures, *Australasian Telecommunication Networks and Applications Conference (ATNAC 2007)*, pp. 197–202.

Stevens, W. R. (1994). *TCP/IP illustrated (vol. 1): the protocols*, Boston:Addison-Wesley, chapter 19.4 Nagle Algorithm, pp. 267–273.

Stevens, W. R. and Wright, G. R. (1995). *TCP/IP illustrated (vol. 2): the implementation*, Boston:Addison Wesley, chapter 25. TCP Timers, pp. 817–850.

Tolia, N., Andersen, D. G. and Satyanarayanan, M. (2006). Quantifying interactive user experience on thin clients, *IEEE Computer* **39**(3): 46–52.

Vankeirsbilck, B., Simoens, P., De Wachter, J., Deboosere, L., De Turck, F., Dhoedt, B. and Demeester, P. (2008). Bandwidth optimization for mobile thin client computing through graphical update caching, *Australasian Telecommunication Networks and Applications Conference (ATNAC 2008)*, pp. 385–390.

Visweswaraiah, V. and Heidemannk, J. (1997). Improving restart of idle TCP connections, *Technical Report 97-661*, University of Southern California.

Wireshark® (2011). wireshark®, `http://www.wireshark.org/`. [accessed 21 March 2011].

Yamanegi, K., Hama, T., Hasegawa, G., Murata, M., Shimonishi, H. and Murase, T. (2005). Implementation experiments of the TCP proxy mechanism, *Proc. of Information and Telecommunication Technologies, 2005. APSITT 2005. 6th Asia-Pacific Symposium*, pp. 17–22.

Yang, S. and Tiow, T. T. (2007). Improving interactive experience of thin client computing by reducing data spikes, *ACIS International Conference on Computer and Information Science*, pp. 627–632.