Performance Analysis and Improvement of TCP Proxy Mechanism in TCP Overlay Networks

Ichinoshin Maki†, Go Hasegawa‡, Masayuki Murata†, Tutomu Murase§
†Graduate School of Information Science and Technology, Osaka University
‡Cybermedia Center, Osaka University
§System Platforms Research Laboratories, NEC corporation
†1-5 Yamadaoka, Suita, Osaka, 565-0871, Japan
‡1-32 Machikaneyama, Toyonaka, Osaka, 560-0043, Japan
§1753, Shimonumabe, Nakahara-ku, Kawasaki, Kanagawa, 211-8666, Japan

Abstract— **TCP overlay networks that control data transmission quality at the transport layer are being paid a lot of attentions as users' demands for diversified Internet services increase. They are expected to enhance the end–to–end throughput of the TCP connection essentially because the round trip times and the packet loss ratios of each split TCP connection are reduced. However, performance degradation may occur due to undesired interactions among the split TCP connections. In this paper, we introduce an analysis approach to estimate end–to–end throughput of data transmission with a TCP proxy mechanism considering of performance degradation. Our analysis results revealed that we confirmed the effect of the TCP proxy mechanism. We also found that we cannot ignore performance degradations due to interactions among split TCP connections especially when the congestion level of the network they traverse is small. Further, we clarified that we should take into account the packet loss ratios, performance degradations and propagation delays of the network when we consider the issue relating to the design of TCP overlay networks.**

I. Introduction

The tremendous Internet development has been greatly spurred by access/backbone network technologies such as xDSL and optical fiber. As well, users' demands for diversified services have increased due to the rapid growth of the Internet population. Some of these applications require high quality transport services in terms of end–to–end throughput, packet loss ratio, delay, and so on. However, data transmission quality across the present Internet cannot be assured, essentially because of its best–effort basis.

IntServ [1] and DiffServ [2] are possible solutions for the problem by adding control mechanisms at the network layer. For example, the Diffserv architecture is based on a simple model where traffic entering a network is classified and possibly conditioned at the boundaries of the network, and then assigned to different behavior aggregates. However, they would be necessary to deploy additional mechanisms to all routers that all traffic–flows traverse in order to provide sufficient benefit from the introduction of IntServ/DiffServ to the network. Therefore, because of aspects such as scalability and cost, we believe that these schemes have almost no chance of being deployed to the large–scale network.

There are other approaches for quality control mechanism, which are located under/over IP layer. MPLS (Multi–Protocol Label Switching) [3] and GMPLS (Generalized MPLS) [**?**] are typical examples of the underlay approach. For example, MPLS allows a particular packet stream to follow a pre–determined path rather than a path computed by hop–by–hop destination based routing. Although these approaches are well performed in the internal of an ISP (Internet Service Provider), they are not applicable to data transmission passing through multiple ISPs; they need additional mechanisms such as bandwidth broker [4]. Therefore, they have the same shortcoming as IntServ/Diffserv architectures in scalability and deployment.

Proxy cache servers in CDNs (Contents Delivery Networks) [5] and media streaming in P2P (Peer to Peer) network [6] are typical examples for overlay networking approach. In overlay networks, packets from a sender host are forwarded to a receiver host via some other hosts/nodes which exist there. The route between the sender and receiver hosts these packets traverse is composed of many virtual paths in the overlay networks. This means that overlay networks can provide various services without changes to the existing IP infrastructure even if overlay networks spread over multiple ISPs. The overlay networks control data transmission quality by using information of the underlying IP network by means of monitoring and/or signaling mechanisms. For example, application–layer overlays in [7-9] use RTTs (Round Trip Times) and hop–counts between overlay nodes in order to configure the topology of the overlay networks and select adequate paths between sender and receiver nodes. The other researches on the overlay network for IP packet routing, such as RON (Resilient Overlay Network) [10] and FBR (Feedback Based Routing) [11], obtain the functioning and quality of the Internet paths among overlay nodes and use this information to decide whether to route packets directly over the Internet or by way of other overlay nodes. However, these overlay schemes need additional overheads such as signaling messages and redundant traffic for measuring the network performance and exchanging information among overlay nodes. Another disadvantage is that they need some complicated control mechanisms specific to each application, and that parameter settings are very sensitive to various network factors.

In this paper, we introduce a TCP overlay network as a scalable and deployable overlay network. It controls data transmission qualities at the transport layer, meaning that the IP layer remains providing only minimum fundamental functions such as the routing and packet forwarding. It is fundamentally different from IntServ/DiffServ that requires sophisticated facilities to the network layer and RON/FBR that needs additional overheads for measuring the network performance. One of the important mechanisms of TCP overlay networks is to divide the end–to–end TCP connection into multiple split TCP connections, as shown in Fig. 1, to control transmission quality at the transport layer. In this paper, we call this splitting mechanism *TCP Proxy*.

The advantage of TCP overlay networks is the improvement of TCP throughput achieved by TCP proxy, where data packets are relayed from the sender host to the receiver host via the split TCP connections. Since the shorter TCP loops enable us to realize a shorter RTT and lower packet loss ratio, the TCP throughput can be increased by TCP proxy. Furthermore, the shorter RTT and lower packet loss ratio also make it easy to control the performance of TCP connections. For example, by introducing the TCP proxy mechanism, differences in the network environment can be concealed from users; if the network between sender and receiver hosts includes a wireless network, the end–to–end throughput of the TCP connection generally deteriorates due to the high packet loss ratio and large propagation delay within the wireless network. In this case, performance degradation can be minimized by splitting the TCP connection at both the ingress and egress edges of the wireless network. Then, data transfer in the wireless network becomes isolated from that of other parts of the network, and vice versa.

By using the TCP overlay network for data transmission, the upper–layer application by itself does not need to decide paths among overlay nodes. The paths are selected by TCP proxy

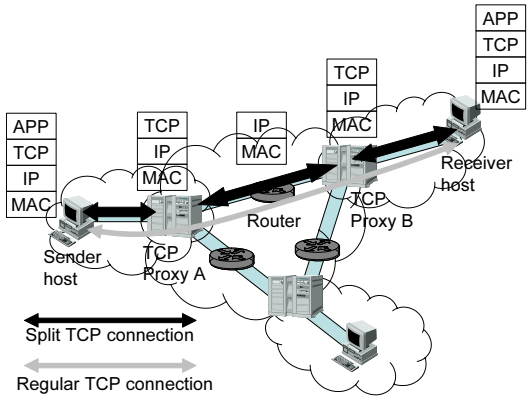


Fig. 1. TCP overlay network.

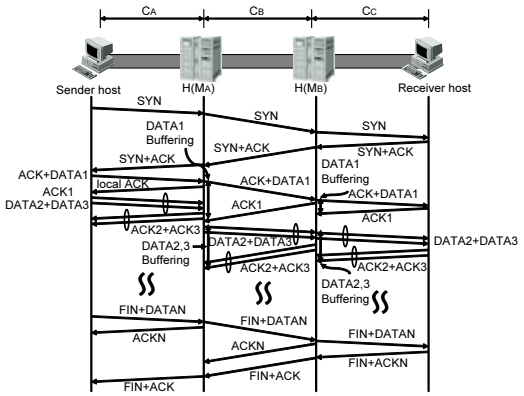


Fig. 2. TCP proxy mechanism.

nodes, using information obtained from TCP connections between TCP proxy nodes. Therefore, the application can omit annoying tasks such as the measurement of IP network. Furthermore, it is expected that the application exposes good performance by obtaining better information from the transport layer. TCP throughput is a typical example which can not be obtained by the approaches in [10, 11], and we consider it would reflect the performance of the upper–layer application.

TCP overlay networks has quite a high applicability to the deployable network. That is, it is not necessary to introduce the TCP proxy mechanism to all routers in the network, while IntServ/DiffServ needs to deploy additional mechanisms to all routers, due to the end–to–end principle of TCP. The advantage of the TCP proxy mechanism can be exhibited even when only one proxy exists in the network, and the larger the number of the TCP proxy nodes becomes, the larger performance gain can be obtained. It means that the data transfers traversing multiple ISPs can be easily enhanced by a TCP proxy mechanism. Furthermore, there is no need to modify the end user's protocol stack, since our TCP proxy mechanism automatically splits the user's TCP connection.

The idea of TCP proxy is not a new idea. In previous reports [**?**12-15], some schemes have been proposed to improve data transfer throughput by splitting TCP connections. Some researches have focused on specific networks such as wireless and satellite [12-14]. Other reports have clarified the advantage of improvements of data transfer throughput [**?**15] but do not take account of the serious problems involved in splitting TCP connections and relaying data packets. However, we believe that we should not limit to apply to TCP proxy mechanism only to specific networks, and that the merit of TCP overlay networks described above becomes apparent when we consider its deployment to general networks. In fact we cannot expect that the TCP proxy mechanism will provide the drastic improvement in end–to–end throughput described in some previous reports. As we will discuss later, various kinds of performance degradations may occur in splitting TCP connections, due to undesired interactions among the split TCP connections. Those problems become more obvious when we try to minimize the degree of modification of the current system in introducing the TCP proxy mechanism.

In this paper, we introduce an analysis approach to estimate the end–to–end throughput of data transmission with a TCP proxy mechanism. We take into account the problems that may occur in introducing the TCP proxy mechanism on the analysis approach. From this analysis, we confirm the effect of the TCP proxy mechanism in various kinds of the networks, and ascertain the degree of the performance degradation. Furthermore, by using the analysis results we show a simple solution to performance degradation, which sets the proper size of the send/receive socket buffer size. Finally, we discuss the issue of the design of TCP overlay networks with a TCP proxy mechanism.

The rest of this paper is organized as follows. Section II describes the TCP proxy mechanism that is a fundamental mechanism in TCP overlay networks and shows a very rough estimation of the effect of TCP proxy without any performance degradation problems. We then point out some problems related with the TCP proxy mechanism. Section III describes our new analysis approach to estimate end–to–end throughput of data transmission with the TCP proxy mechanism considering performance degradation. Section IV discusses the effectiveness of the TCP proxy mechanism using our analysis results, the degree of performance degradation, and the issue of the design of TCP overlay networks. Finally, in Section V we present our conclusions and note future works.

II. TCP proxy mechanism

A. Overview

In TCP overlay networks, a TCP proxy is a fundamental mechanism which splits a TCP connection between sender and receiver hosts into multiple TCP connections at some network nodes. Fig. 2 illustrates how data packets are transferred from a sender host to a receiver host by split TCP connections when a TCP connection is divided at both TCP proxy A ($H(M_A)$) and B ($H(M_B)$) as shown in Fig. 1. Here, we define each split TCP connection as C_A, C_B, and C_C from the sender host. When a packet from the sender host arrives at $H(M_A)$ via C_A, $H(M_A)$ relays them to C_B. Similarly, $H(M_B)$ relays packets from C_B to C_C. In the TCP overlay networks discussed in this paper, we use a local ACK packet [16]; a TCP proxy node sends back a pseudo ACK packet to the upward sender/proxy when it receives a data packet, without waiting to receive an ACK packet from the downward receiver/proxy, as shown in Fig. 2. By using local ACK packets, the sender host can transfer new data packets without waiting for ACK packets to be received from the receiver host. This is expected to improve data transfer throughput of the connection C_A by shortening the RTT value. Similarly, the throughputs of both C_B and C_C are also improved, which results in the improvement of end–to–end throughput between sender and receiver hosts. Furthermore, a TCP proxy has send/receive socket buffers for storing data packets, just as a regular TCP host does. Therefore, when a data packet is lost between $H(M_B)$ and the receiver host, C_C can retransmit the dropped packets from $H(M_B)$ instead of the sender host. It is also expected to improve data transfer performance compared to that of a regular TCP connection.

In a strict sense, this proxy mechanism violates TCP semantics. That is, by using local ACK packets, the sender host receives ACK packets for data packets before the receiver host receives them, which may deteriorate reliable data transmission of TCP. However, we believe that reliability can be maintained since each split TCP connection has the same reliability as a regular TCP connection. Another reason is that a TCP proxy node forwards SYN and FIN packets, which are used in connection establishing and termination, in an normal manner without using local ACK packets. In TCP overlay networks, splitting is performed at TCP proxy nodes where both SYN and

TABLE I
THE EFFECTIVENESS OF TCP PROXY MECHANISM.

Number of Split Connections	Normalized Throughput
2	2.823370
4	7.977354
8	22.550680
16	63.764141
32	180.320358

SYN/ACK packets are passed through. Even if packets from a sender host are forwarded to a receiver host in different routes, a TCP proxy mechanism operates successfully because data transfer is performed via split TCP connections among TCP proxy nodes. However, we assume that all packets invariably traverse the edge proxy nodes, which are the nearest proxy nodes from the sender/receiver hosts. This is because we want not to modify the sender/receiver TCP implementations. We consider the following two methods to overcome this problem; one is that the first and last hop TCP proxy nodes should be located at the focal points of the network. The other is that a TCP proxy software is installed to TCP sender/receiver hosts, so that the TCP sender/receiver hosts virtually behave the first and last hop TCP proxy nodes.

B. Simple throughput estimation

As stated above, improvements of data transfer throughput are expected from the introduction of the TCP proxy mechanism. This is because a TCP connection is divided into multiple split TCP connections and each split connection forwards packets with shorter control loops. We therefore consider that the expected end–to–end throughput ρ can be calculated as follows.

$$\rho = \min_i \rho(i)$$

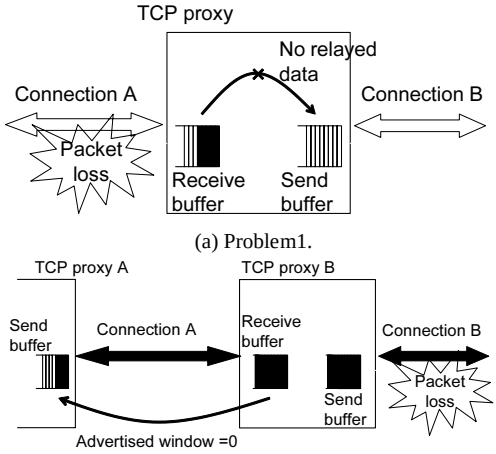
This expression shows that end–to–end throughput equals the smallest throughput $\rho(i)$ of each split TCP connection i. The average throughput ρ of a TCP connection can be estimated by using Eq.(1), which is described in [17]:

$$\rho = \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_o \cdot min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)} \equiv \mathrm{TCP\_RHO}(RTT, p, T_o, b) \quad (1)$$

Here, we denote b as the option of the delayed ACK, where the receiver host replies to one ACK packet every b ACK packets. p, RTT, and T_o are the packet loss ratio, round trip time, and time duration of the initial timeout, respectively. Table I shows the effectiveness of the TCP proxy mechanism in a 32 hop network using the above estimation. In this table, the first row is the number of split TCP connections, each of which has identical hop counts; and the second row is the normalized throughput, which is defined as the ratio of the throughput when using TCP proxies to that without TCP proxies. We also set the link bandwidths, the packet loss ratios and the propagation delays of each hop as 100 [Mbps], 0.0005 and 0.01 [s], respectively. From this table, we can observe that end–to–end throughput is greatly improved as the number of split TCP connections becomes large. This is because both of the RTTs and packet loss ratios of each split TCP connection become small.

C. Problems in TCP proxy mechanism

The above calculation is for an ideal case, where we do not consider any bad effects from splitting the TCP connections. In a practical case, however, some performance degradations may occur due to undesired interactions among the split TCP connections. Here, we introduce two major problems causing degradation of the data transfer throughput, especially in regard to buffering at the TCP proxy nodes. Fig. 3(a) illustrates one problem caused by temporary congestion at a split TCP connection A. When a packet loss occurs at the split TCP connection A due to network congestion, the TCP proxy temporarily stops relaying data packets from the receive buffer of the split TCP connection



(a) Problem1.

(b) Problem2.

Fig. 3. Two problems in the TCP proxy mechanism.

A to the send buffer of the split TCP connection B until the lost packet is retransmitted and arrives at the proxy. This is because a TCP receiver deals with the received data in an in–order fashion. Then, the send buffer of connection B may become empty since connection B continues sending packets. As a result, the throughput of connection B deteriorates. This problem does not occur when we do not use a TCP proxy since all incoming packets are immediately forwarded by the normal router regardless of the order of the arriving packets.

Fig. 3(b) depicts another problem where the throughput of the split TCP connection B deteriorates when it experiences network congestion. We consider the situation where connection B temporarily cannot send data packets from the send buffer at the TCP proxy B due to network congestion. When the duration of the congestion is long, the buffers at the TCP proxy B (the send buffer for connection B and the receive buffer for connection A) become full since split TCP connection A continues transmitting packets to the TCP proxy B. When there is no remaining space in the receive buffer, TCP proxy B sends local ACK packets with zero size of the advertised window. This causes connection A to stop sending data packets for a while. Then, the buffers at proxy B become empty when the connection B recovers from the congestion and starts sending packets again. Therefore, the throughput of connection B may deteriorate.

Those problems are caused by the introduction of a TCP proxy mechanism in the network, meaning that we put TCP endpoints in the network routers, in addition to the sender/receiver hosts. Therefore, we should take them into account when evaluating the performance of the TCP overlay network. However, we consider the problem shown in Fig. 3(b) is not so serious in an actual network. This is because major operating systems including Linux and FreeBSD have a mechanism to avoid performance degradation from a zero advertised window [18, 19]; when the received data is retrieved from the receive socket buffer to the application buffer and there becomes some available space in the receive buffer, the receiver–side TCP sends an additional ACK packet to the sender to inform the new value of the advertised window. In the next section, we consider the problem in Fig. 3(a) as the major reason for performance degradation in splitting TCP connections.

III. Throughput analysis

A. Model and assumptions

We use the network model as shown in Fig. 4 in our analysis. We focus on a TCP connection which traverses $n+1$ nodes from the sender host $H(0)$ to the receiver host $H(n)$. We define link bandwidth, packet loss ratio, and propagation delay of a link as $L(h)(1 \leq h \leq n)$ as $b(h)$, $p(h)$, and $d(h)$, respectively. We assume that the TCP connection is divided into m split TCP connections at $m-1$ TCP proxy nodes. A split TCP

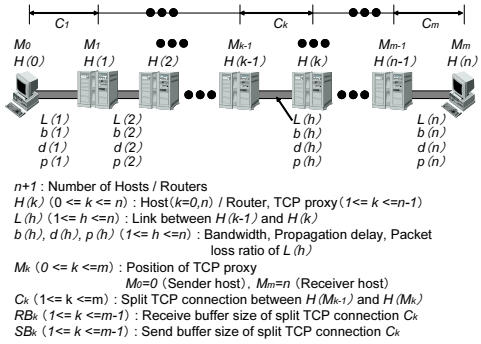


Fig. 4. Analysis model.

connection $C_k (1 \leq k \leq m)$ is established between the TCP proxy $H(M_{k-1})$ and the TCP proxy $H(M_k)$, where M_k is the position of the TCP proxy. $H(M_k)$ has the receive socket buffer for C_k, the size of which is RB_k, and the send socket buffer for C_{k+1}, the size of which is SB_{k+1}. The goal of our analysis is to estimate the average end–to–end throughput ρ of the data transfer from $H(0)$ to $H(n)$, considering the performance problem depicted in Fig. 3(a). Since our analysis is based on the analysis of the average throughput of a TCP connection in [17], we use the same assumptions as those in [17]. We further introduce a new assumption that a TCP proxy sends a local ACK packet when it receives a data packet from the upward sender/proxy. We do not consider a processing overhead at TCP proxy nodes because we are interested in the average end–to–end throughput in this paper. However, the processing overhead should be taken into account especially when we evaluate transfer delays of the fixed–sized data. We are now investigating the effect of processing overhead and we will show the results in the final–version paper.

B. Analysis

Our analysis makes an iterative calculation. First, we calculate the average throughput $\rho_k[0]$ of C_k without consideration of the interaction among split TCP connections. That is, $\rho_k[0]$ is the average throughput of C_k where we do not consider the interaction between C_k and C_{k-1}, and between C_k and C_{k+1}. Then, we define $\rho[0] = \min_{1 \leq k \leq m} \rho_k[0]$, which is the first value of the iterative calculation. In the i–th iteration, we calculate $\rho_k[i]$ and $\rho[i]$ based on $\rho_k[0]$, $\rho_k[i-1]$ and $\rho[i-1]$ considering the performance degradation problem. The iteration stops when the following condition becomes satisfied, and then we consider $\rho[i]$ as the average throughput ρ.

$$\frac{|\rho[i-1] - \rho[i]|}{\rho[i-1]} < \epsilon \tag{2}$$

$\rho_k[0]$ is determined by three factors; (A) the throughput given by using Eq.(1), (B) the bandwidth–delay product of the network between $H(M_{k-1})$ and $H(M_k)$, and (C) the receive buffer size of C_k, RB_k . In the case of (A), the average throughput of C_k is derived from Eq.(1) as follows:

$$\rho_k^{'}[0] = TCP\_RHO(RTT_k, p_k, T_o, b)$$

Here, T_o, p_k (packet loss ratio of C_k) and RTT_k (RTT of C_k) are calculated as follows:

$$T_o = 4 \cdot RTT_k \tag{3}$$

$$\begin{aligned} p_k &= 1 - (1 - p(M_{k-1}+1)) \cdot (1 - p(M_{k-1}+2)) \cdot \\ &\quad \cdots (1 - p(M_k)) \\ &= 1 - \prod_{h=1}^{M_k - M_{k-1}} (1 - p(M_{k-1}+h)) \end{aligned}$$

$$RTT_k = 2 \sum_{h=M_{k-1}+1}^{M_k} d(h)$$

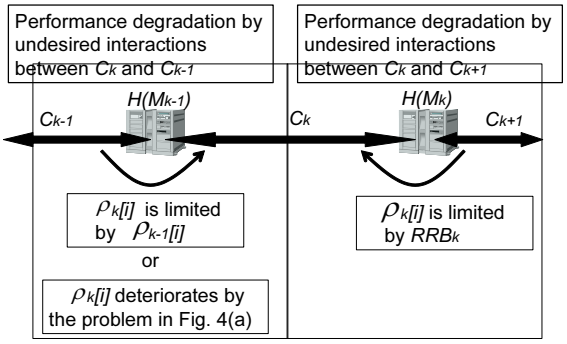


Fig. 5. Performance degradation by undesired interactions among split TCP connections.

Eq.(3) is a very rough estimation of RTO (Retransmission Time-Out) presented in [20]. In the case of (B), the average throughput of C_k equals the minimum link bandwidth of the traversing network;

$$\rho_k^{''}[0] = \min_{M_{k-1}+1 \leq h \leq M_k} b(h)$$

In the case of (C), it is calculated as follows:

$$\rho_k^{'''}[0] = \frac{RB_k}{RTT_k}$$

Therefore, $\rho_k[0]$ and $\rho[0]$ in the first iteration are calculated as follows:

$$\rho_k[0] = min\left(\rho_k^{'}[0], \rho_k^{''}[0], \rho_k^{'''}[0]\right)$$

$$\rho[0] = \min_{1 \leq k \leq m} \rho_k[0] \tag{4}$$

We then calculate the average throughput of the i–th iteration. We calculate $\rho_k[i]$, the throughput of split TCP connections C_1, C_2, $\cdots$, C_m in this order, and we assume that $\rho_m[i]$ equals $\rho[i]$. When we take into account the interactions among split TCP connections, we should consider the remaining space in the receive buffer, which depends on the throughputs of both C_k and C_{k+1}. Here, we model packet arrival/departure at/from the receive buffer as M/M/1/K queuing model, where $K = RB_k$. We consider $\rho[i-1]$ as the average packet arrival rate at the receive buffer of C_1 and $\rho_2[0]$ as the average service rate in the send buffer of C_2. The remaining space RRB_1 in the receive buffer of C_1 is then calculated as follows:

$$RRB_1 = min\left(RB_1, K - \frac{\rho_1^{'}\{1-(K+1)\rho_1^{'K} + K\rho_1^{'K+1}\}}{(1-\rho_1^{'})(1-\rho_1^{'K+1})}\right)$$

where $K = RB_1 + SB_2$ and $\rho_1^{'} = \rho[i-1]/\rho_2[0]$. Therefore, we can calculate the average throughput of C_1 as follows:

$$\begin{aligned} \rho_1[i] &= min\Bigg(\mathrm{TCP\_RHO}(RTT_1, p_1, T_o, b), \\ &\qquad \min_{M_0+1 \leq h \leq M_1} \left(b(h), \frac{RRB_1}{RTT_1}\right)\Bigg) \end{aligned}$$

In order to calculate $\rho_k[i]$ $(2 \leq k \leq m)$, we take into account undesired interactions between C_k and C_{k-1}, and between C_k and C_{k+1}. See Fig. 5. We consider the time duration between two successive packet drops in C_{k-1}, which we define as one cycle. We denote $p_k(j)$ as the probability that the number of packets stored in the send buffer of C_k is j $(0 \leq j \leq SB_k)$ at the beginning of the cycle. We also denote $\rho_k\{j\}$ as the average throughput of C_k when there are j packets in the send buffer of C_k. We can therefore calculate the average throughput of C_k in the i–th iterative calculation;

$$\rho_k^{''''}[i] = \sum_{j=0}^{SB_k} p_k(j) \cdot \rho_k\{j\} \tag{5}$$

In what follows, we explain the derivation of $p_k(j)$ and $\rho_k\{j\}$. We model the behavior in the send buffer of C_k as M/M/1/K

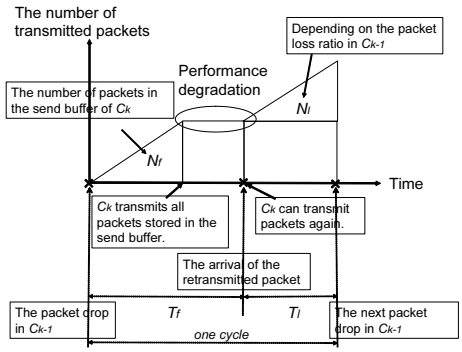


Fig. 6. Performance degradation as shown in Fig. 3(a).

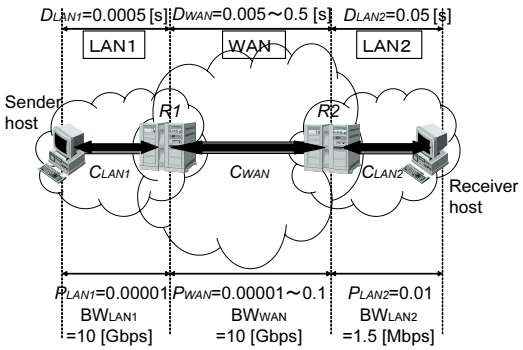


Fig. 7. Simulation model.

queue to calculate $p_k(j)$. We consider $\rho_{k-1}[i]$ as the average packet arrival rate to the send buffer of C_k and $\rho_k[0]$ as the average service rate in the send buffer of C_k. $p_k(j)$ is then calculated as follows:

$$p_k(j) = \begin{cases} \dfrac{\rho_k^{''j}(1-\rho_k^{''})}{1-\rho_k^{''K+1}} & \rho_k^{''} \neq 1 \\ \dfrac{1}{K+1} & \rho_k^{''} = 1 \end{cases}$$

where $K = SB_k$ and $\rho_k^{''} = \rho_{k-1}[i]/\rho_k[0]$. The packet arrival rate at the send buffer of $C_k (2 \leq k \leq m)$ depends on the throughput of an upward connection in current iteration while the packet arrival rate at the send buffer of C_1 does not so. We therefore use $\rho_{k-1}[i]$, which is different from the packet arriva rate $\rho[i-1]$ at the send buffer of C_1, as the packet arrival rate at the send buffer of C_k. We consider the cycle is divided into two parts; the time from the beginning of the cycle to the arrival of the retransmitted packet at $H(M_{k-1})$, and the time from that to the end of the cycle. We denote T_f as the average time duration of the former part and N_f as the average number of packets which C_k transmits in T_f. T_l and N_l are also defined as those of the latter part. If there are remaining packets in the send buffer when the retransmitted packet arrives, no performance degradation occurs as shown in Fig. 6. Therefore, we can calculate $\rho_k\{j\}$ in that case as follows:

$$\rho_k\{j\} = \rho_k[0] \quad (T_f \cdot \rho_k[0] < j)$$

Otherwise, we can calculate it as follows:

$$\rho_k\{j\} = \frac{N_f + N_l}{T_f + T_l} \tag{6}$$

In the following, we discuss the derivation of N_f, T_f, N_l and T_l by using Fig. 6. N_f is the number of packets stored in the send buffer of C_k when a packet loss occurs in C_{k-1}. Therefore, N_f is calculated as follows:

$$N_f = j$$

T_f is the time duration for the retransmission of the lost packet. It depends on whether the retransmission is caused by fast retransmit or timeout in TCP mechanism [21]. It is a reasonable assumption that it takes about RTT_{k-1} in fast retransmit, and RTO (Retransmission TimeOut) in timeout. Therefore, T_f is calculated as follows:

$$\begin{aligned} T_f &= \left(1 - min\left(1, \frac{3}{w(k-1)}\right)\right) RTT_{k-1} \\ &\quad + min\left(1, \frac{3}{w(k-1)}\right)\left(T_o + \frac{1}{2}RTT_{k-1}\right) \end{aligned}$$

Here, we denote $w(k-1)$ as the average window size of C_k when a packet loss occurs and $min(1, 3/w(k-1))$ is the probability that the lost packet is detected by the timeout. These two values can be found in [17]. N_l is the average number of packets which C_k transmits between two successive packet drops. Then,

$$N_l = \frac{1}{p_{k-1}}$$

T_l is the time duration for C_k to transmit N_l packets. It is assumed that when the retransmitted packet arrives at $H(M_{k-1})$, C_k has enough packets in the send buffer. We therefore can calculate T_l as follows:

$$T_l = \frac{\dfrac{1}{p_{k-1}}}{min(\rho_{k-1}[i], \rho_k[0])}$$

By using these N_f, T_f, N_l and T_l, we can calculate $\rho_k\{j\}$ in Eq.(6). We then obtain $\rho_k^{''''}[i]$ in Eq.(5).

Furthermore, $\rho_k[i]$ may be limited by RRB_k, the bandwidth–delay product of the networks, as is in the case of C_1. Then, $\rho_k[i]$ is calculated as follows:

$$RRB_k = min\left(RB_k, K - \frac{\rho_k^{'}\{1-(K+1)\rho_k^{'K}+K\rho_k^{'K+1}\}}{(1-\rho_k^{'})(1-\rho_k^{'K+1})}\right)$$

$$\rho_k[i] = min\left(\rho_{k-1}[i], \rho_k^{''''}[i], \min_{M_{k-1}+1 \leq h \leq M_k}\left(b(h), \frac{RRB_k}{RTT_k}\right)\right)$$

where $K = RB_k + SB_{k+1}$ and $\rho_k^{'} = \rho[i-1]/\rho_{k+1}[0]$. We then obtain end–to–end throughput $\rho[i]$ of the i–th iteration, considering the performance degradation that occurs in introducing the TCP proxy mechanism as follows.

$$\rho[i] = \rho_m[i]$$

We continue this iterative calculations until Eq.(2) is satisfied, and finally derive the analysis results ρ as $\rho[i]$ when the iteration stops.

C. Numerical examples

In this subsection, we confirm the correctness of our analysis approach by comparing the simulation results. All simulations were run using NS simulator [22]. We compare analysis results to simulation results in some cases. All results show the correctness of our analysis approach. We do not show all results because of the space limitation. In this paper, we investigate the basic performance of the TCP overlay network by using a small simulation model. However, it is qualitatively obvious that the TCP overlay network shows good performance in a large scale network. In future investigations, we need to evaluate the performance of the TCP overlay network in such a case.

We use the network topology shown in Fig. 7. The network topology is composed of three networks; LAN1, WAN and LAN2. The TCP proxy is deployed at the edge of each network and a TCP connection between sender and receiver hosts is divided into three split TCP connections (C_{LAN1}, C_{WAN}, C_{LAN2}) at two TCP proxies on $R1$ and $R2$. We used TCP Reno version which is the most popular in the Internet. Other versions of TCP such as SACK, NewReno, and Vegas can be applicable in a TCP proxy mechanism, but the degree of the performance gain remains almost the same since it is independent of the details of the congestion control algorithm of TCP. We denote D_{LAN1}, D_{WAN}, D_{LAN2} as the propagation delay, P_{LAN1}, P_{WAN}, P_{LAN2} as the packet loss ratio, BW_{LAN1}, BW_{WAN}, BW_{LAN2} as the link bandwidth of each network. BW_{LAN1}, BW_{WAN} and BW_{LAN2} are enough to not limit the performance

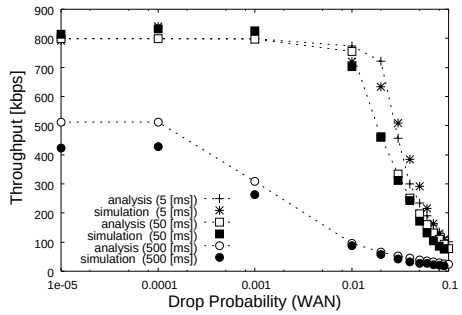


Fig. 8. Confirmation of the analysis results.

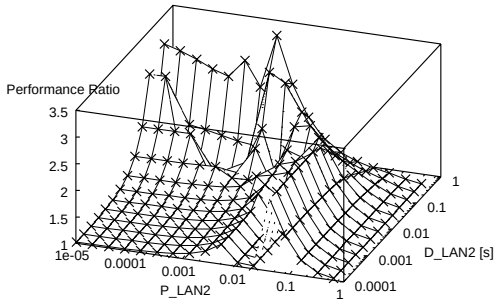


Fig. 9. The effectiveness in introducing a TCP proxy mechanism.

of the TCP connection. We use 0.01 as the value of ϵ in Eq.(2). Note that the other values of ϵ make no difference in the simulation results. In the following numerical examples, we show the results when the packet loss ratio and propagation delay between a TCP proxy on $R2$ and the receiver host is large. It means that we assume that users access the Internet via satellite or wireless networks. Note that the correctness of our analysis approach is proved to be almost identical in other cases.

Fig. 8 shows end–to–end throughput when D_{WAN} is 5, 50, 500 [ms]. In this figure, the achievable throughput is very high when D_{WAN} is 5 and 50 [ms] and P_{WAN} is less than 0.01. In these regions, we can find that throughput equals that of C_{LAN2}, although the throughput of C_{LAN2} is the smallest among the three connections. This confirms the effective of the TCP proxy, which eliminates the adverse impact caused by an increase of P_{WAN}. On the other hand, when P_{WAN} is larger than 0.01, the throughput of C_{WAN} is the smallest among the three connections. Therefore, the larger P_{WAN} is, the smaller the end–to–end throughput is. This phenomenon also occurs when P_{WAN} is larger than 0.0001 and D_{WAN} is 500 [ms]. From this figure, the analysis results give a reasonable estimation of end–to–end throughput. However, when D_{WAN} is small and P_{WAN} is large, and D_{WAN} is large and P_{WAN} is small, there is a significant difference between analysis and simulation results. From our investigation, this difference is not caused by our analysis, but by the analysis in [17]. The authors in [17] said that the accuracy of the analysis in [17] is not assured when packet loss ratio is less than 0.001 or RTT value is less than 0.1 sec. Another reason is that the analysis in [17] does not consider the time spent in slow start phase, it cannot give good throughput estimation when packet loss ratio is large and/or RTT value is relatively large.

IV. Performance evaluation

In this section, we examine the performance of the TCP proxy mechanism using the analysis results, and investigate its effectiveness. We first discuss performance gain obtained by introducing the TCP proxy mechanism. We then discuss the degree of performance degradation caused by the problems described in Subsection II-C, and the network characteristics where the performance degradation is large. We next detail a simple solution to avoid performance degradation and show its effectiveness. Finally, we discuss the issue of the design of TCP overlay networks with a TCP proxy mechanism from the point of view of performance.

We use the network model depicted in Fig. 7 and set BW_{LAN2} to 10 [Gbps]. We compare the following two cases; one is when the end–to–end connection is established between sender and receiver hosts (case 1), the other is when one end–to–end connection is split at two proxies (case 2). In case 1, the average end–to–end throughput is calculated by using Eq.(1). In case 2, it is calculated by using the proposed analysis methods described in Section III and performance degradation depicted in Fig. 3(a) occurs. Fig. 9 shows the change of performance ratio when P_{LAN2} and D_{LAN2} are set to various values. Here, we define the performance ratio as the ratio of average end–to–end throughput in case 2 to that in case 1. From this figure, we can observe that the ratio is always larger than 1 independent of the values of P_{LAN2} and D_{LAN2} and the degree of performance improvement is up to about 3 times. It means that a TCP proxy mechanism is very useful even if we consider performance degradation among split TCP connections.

In order to examine the characteristics of performance degradation, we use the same network topology as in Section III, where we set P_{LAN1} to 0.01, D_{LAN1} to 0.05 [s], P_{WAN} to 0.01 and D_{WAN} to 0.05 [s]. Fig. 10 shows the ratio of the performance degradation as a function of P_{LAN2} and D_{LAN2}. Here, the ratio is defined as the throughput obtained in the analysis in Section III, divided by the expected throughput shown in Eq.(4) without performance degradation. From this figure, when P_{LAN2} is about 0.01 and D_{LAN2} is about 0.05, performance degradation tends to become large. That is, when the difference of the network congestion level between WAN and LAN2 is small, there is serious performance degradation. In such a situation, since the average number of packets stored in the send buffer of the split TCP connection is small, performance degradation frequently takes place, as explained in Section III. The degree of performance degradation is up to about 40% of the expected throughput in this example, which means that we cannot ignore problems caused by splitting the TCP connection.

As explained above, performance degradation occurs since the number of packets stored in the send buffer of the TCP proxy becomes zero. Therefore, one may expect that we can avoid performance degradation by enlarging the send buffer size. To show effectiveness of this, we show another simulation result where we again use the simulation model depicted in Fig. 7 and P_{WAN} and P_{LAN2} are 0.01, and D_{WAN} and D_{LAN2} are 0.05, respectively. These parameters are the values when performance degradation is maximum in Fig. 10. Fig. 11 shows the ratio of performance degradation to the expected throughput as functions of send/receive buffer sizes at TCP proxy on $R2$. From this figure, we find that performance degradation becomes smaller by enlarging the send buffer size. This is because the probability that the number of packets stored in the send buffer is zero becomes small. In this example, when we do not consider performance degradation, end–to–end throughput should become about 800 Kbps, corresponding to 1.0 of the y–axis in Fig. 11. To obtain this throughput value, a TCP connection requires 10 [KBytes] for its send/receive buffer size. This value is relatively small compared with the default values assigned to send/receive buffers in the currently operating systems. That is, we can expect 800 [kbps] throughput when we do not consider the performance degradation with a TCP proxy. However, Fig. 11 tells us that a TCP proxy needs to allocate more than about 512 [Kbytes] for the send buffer to alleviate throughput degradation. On the other hand, we can also find from Fig. 11 that the receive socket buffer size has little effect on performance. Therefore, we set the receive socket buffer size based on the bandwidth–delay product of the traversing network.

Finally, we discuss the appropriate position of the TCP proxy using the network topology in Fig. 12. We consider that a TCP connection between sender and receiver hosts is divided once at TCP proxies on $R1$, $R2$ or $R3$. The second row of Table II shows the average end–to–end throughput when the

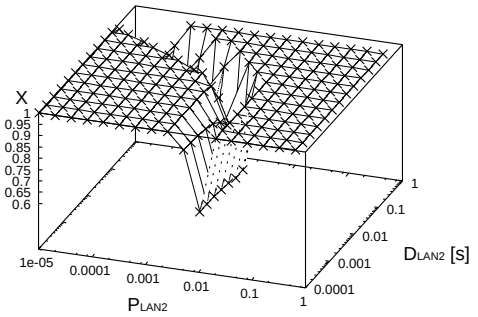


Fig. 10. The Ratio of performance degradation to expected throughput.

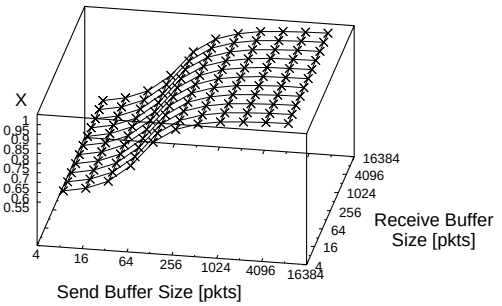


Fig. 11. The Effectiveness of larger buffer size.

network congestion levels of each network is homogeneous ($P_{LAN1} = P_{WAN1} = P_{WAN2} = P_{LAN2} = 0.00001$, $D_{LAN1} = D_{WAN1} = D_{WAN2} = D_{LAN2} = 0.005\ [s]$). From this row, we find that we obtain the largest throughput when the TCP connection is divided at TCP proxy on $R2$. This means that we should divide the TCP connection so that the RTTs of the split connections become equal. The third row of Table II shows the average throughput when the network congestion level is heterogeneous ($P_{LAN1} = P_{WAN1} = P_{WAN2} = 0.00001$, $P_{LAN2} = 0.01$, $D_{LAN1} = D_{WAN1} = D_{WAN2} = D_{LAN2} = 0.005[s]$). From this row, we find that we cannot achieve the largest throughput when we divide the TCP connection based on RTT values of the network. This is because the throughput is greatly affected on that of the split TCP connection between a TCP proxy on $R2$ and the receiver host, where the packet loss ratio is very high. In this case, the throughput becomes largest when the TCP connection is divided at TCP proxy on $R3$ so that the congested network is isolated from the other networks. From these results, we conclude that we must take account of not only the RTTs of each split TCP connection but also packet loss ratios and performance degradations caused by undesired interactions from the introduction of the TCP proxy.

V. Conclusions and future works

In this paper, we initially pointed out the problems that will occur in introducing the TCP proxy mechanism. We have then introduced an analysis approach to estimate end–to–end throughput of data transmission with a TCP proxy mechanism considering these various problems. From our analysis results, we confirmed the effect of the TCP proxy mechanism. Furthermore, we found that we cannot ignore performance degradation caused by these problems, especially when the congestion level of the network where the split TCP connections traverse is small. We have shown that performance degradation can be minimized by enlarging the send buffer size of the split TCP connection, and that the required size is much larger than the size for a normal end–to–end TCP connection. Finally, we clarified that we need to take account of packet loss ratios and performance degradation, as presented in this paper, as well as propagation delays within the network.

In future investigations, we need to evaluate the performance of the TCP proxy mechanism considering the handling of multiple TCP connections, focusing on the processing overhead of incoming packets, and so on. We also need to investigate the performance of the TCP proxy mechanism when it handles Web traffic, where its file transfer delay is severely affected by the processing delays of a TCP proxy. Further, we intend to discuss issues relating to the design of TCP overlay networks in large scaled networks.

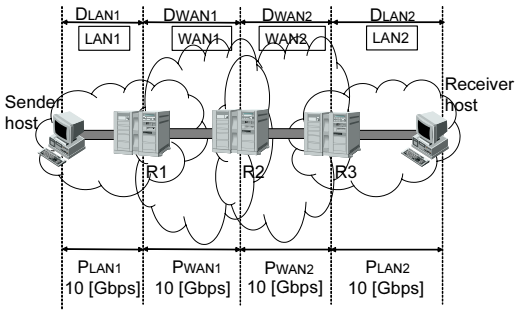


Fig. 12. Simulation model 2.

TABLE II
EFFECT OF THE TCP PROXY IN THE CASE OF A HETEROGENEOUS/HOMOGENEOUS NETWORK.

Split Point	Homogeneous Case	Heterogeneous Case
R1	59.494 [Mbps]	1.860m [Mbps]
R2	108.53 [Mbps]	2.299 [Mbps]
R3	59.494 [Mbps]	3.008 [Mbps]

References

[1] J. Wroclawski, "The use of RSVP with IETF integrated services," *RFC 2210*, Sept. 1997.

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated service," *RFC 2475*, Dec. 1998.

[3] E. C. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," *RFC 3448*, Jan. 2001.

[4] R. Bless and K. Wehrle, "IP multicast in differentiated services networks," *IETF Internet Draft draft-bless-diffserv-multicast-00.txt*, Nov. 2000.

[5] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox, "Removal policies in network caches for World–Wide Web documents," in *Proceedings of ACM SIGCOMM'96*, 1996.

[6] D. A. Tran, K. A. Hua, and T. Do, "ZIGZAG: An efficient peer-to-peer scheme for media streaming," in *Proceedings of IEEE INFOCOM'03*, Mar. 2003.

[7] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault–tolerant wide–area location and routing," tech. rep., Technical Report UCB/CSD-01-1141 UC Berkeley, Apr. 2001.

[8] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large–scale peer–to–peer systems," *Lecture Notes in Computer Science*, vol. 2218, pp. 329–350, 2001.

[9] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proceedings of IEEE INFOCOM'02*, June 2002.

[10] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of ACM SOSP'01*, Oct. 2001.

[11] D. Zhu, M. Gritter, and D. R. Cheriton, "Feedback based routing," *ACM SIGCOMM Computer Communication Review*, vol. 33, pp. 71–76, Jan. 2003.

[12] A. Barkre and B. Badrinath, "I–TCP: Indirect TCP for mobile hosts," in *Proceedings of 15th International Conference on Distributed Computing Systems*, pp. 136–143, June 1995.

[13] H. Balakrishman, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 756–769, Aug. 1997.

[14] M. Luglio, M. Sanadidi, J. Stepanek, and M. Gerla, "The combined use of on–board PEPs and of efficient schemes to improve TCP performance in a satellite environment," in *Proceedings of First International Conference on Advanced Satellite Mobile Systems*, 2003.

[15] P. Rodriguez, S. Sibal, and O. Spatscheck, "TPOT: translucent proxying of TCP," *Computer Communications*, vol. 24, no. 2, pp. 249–255, 2001.

[16] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance enhancing proxies intended to mitigate link-related degradations," *RFC 3135*, June 2001.

[17] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proceedings of ACM SIGCOMM'98*, pp. 303–314, Sept. 1998.

[18] TCP/IP and L. P. Implementation, *Jon Crowcroft and Iain Phillips and John Crowcroft*. John Wiley and Sons, Inc., 2001.

[19] G. R. Wright and W. R. Stevens, *TCP/IP Illustrated, Volume 2: The Implementation*. Reading, Massachusetts: Addision–Wesley, 1995.

[20] M. Handley, S. Floyd, J. Padhy, and J. Widmer, "TCP friendly rate control (TFRC): Protocol specification," *RFC 3448*, Jan. 2003.

[21] J. Postel, "Transmission control protocol," *RFC 793*, Sept. 1981.

[22] "UCB/LBNL/VINT network simulator - ns (version 2)." available at `http://www-mash.cs.berkeley.edu/ns/`.