Regular Paper

TCP Optimization for Eliminating Duplicate Segments in Congested Networks

Yosuke Tamura,† Misako Takayasu†† and Hideki Takayasu†

We propose a new TCP optimization scheme for improving TCP performance over congested networks. The scheme, which we name LEAP, leaps a data segment for retransmission and transmits a new data segment when ACK segment loss is suspected. LEAP is based on judgment whether retransmission of a data segment is caused by a data segment loss or an ACK segment loss. According to our simulation results LEAP eliminates nearly all duplicated segments in heavily congested networks involving multiple connections. Additionally by avoiding such duplicated segments LEAP drastically decreases the number of total transmitted segments in each connection. Our results show that LEAP improves overall network utilization more than 120% higher than conventional TCP variants under congested networks.

1. Introduction

TCP is a powerful and scalable transfer protocol. Most of major network applications such as web browsing, e-mail, telnet, and ftp use TCP as a transport protocol. In the Internet where multiple flows share a single resource, we cannot escape network congestion. This is because the TCP protocol is designed to reduce its transmit rate when congestion is detected, and it increases the transmission rate until detecting network congestion. In the near future, non-TCP-based networked applications like Video On Demand (VOD), 3D hologram, and PanoramaStream will be available in the Internet. These applications should exhibit similar congestion behavior, if they co-exist with TCP-based applications [7),12)]. In such environment where several kinds of multimedia applications share a single resource, improvement of the TCP performance in congested networks is the key for achieving efficient utilization of the network resource.

There are many research works related to reliable transport protocols, which can be categorized into three topics. The first topic is engineering tuning of TCP [3),10),13),23)], which includes pure modifications and improvements under certain network conditions like high error rate, large bandwidth-delay product, heavily congestion, and so on. The second topic is a proposal of a new protocol [1),4),5),18),20)] as an alternative to TCP. This approach aims to propose a higher performance protocol than TCP from some perspectives like error recovery and congestion control. The third topic is analytic modeling of TCP behavior [15),17)]. A mechanism of TCP congestion control is well acceptable by the Internet society. It is an orderly solution for non-TCP flows to apply TCP compatible mechanism into their flow controls [6)]. Our work can be categorized as the first topic. According to our knowledge, this work is the first work to focus on a duplicate received data segments caused by an ACK segment loss.

In this paper, we investigate TCP dynamics in congested networks where network resource is shared by a lot of flows. According to our simulation results, we found that TCP's ACK segment losses causing unneeded duplicate segments at receiver degrade the performance of network utilization in the Internet. Duplicate segments not only waste network resource but also increase overall network loss ratio. To resolve this problem, we propose new TCP retransmission mechanism called LEAP. The reason why we named the scheme as LEAP is that our scheme leaps a data segment for retransmission and transmits a new data segment when an ACK segment loss is expected. LEAP has the following features:

- Simple optimization for TCP sender.
  Sender side modification of a few source codes is needed to support LEAP algorithm.
- Smart activation for congested networks.
  LEAP is compatible to current TCP implementation including congestion control and retransmission scheme. It improves the performance of network utilization under

† Sony Computer Science Laboratories, Inc.
†† Graduate School of Computational Intelligence and Systems Science, Tokyo Institute of Technology

congested networks. In non-congested networks, LEAP shows the same performance as compared to current TCP.

Observations of actual TCP behavior with applying several congestion levels are presented in Section 2. Section 3 describes the principle of TCP, leading to a discussion of our proposed scheme and its implementation in Sections 4 and 5. Section 6 presents an evaluation of our scheme. Related words and discussions are presented in Section 7, and we conclude in Section 8.

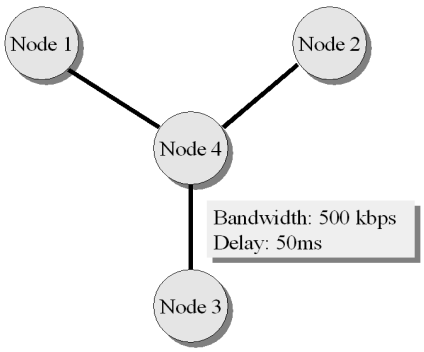

Fig. 1 Simulated network topology.

2. TCP Dynamics

In this section, we investigate TCP dynamics with changing the network congestion level. The results in this paper are based on simulations using the *ns-2*[16] network simulator from Lawrence Berkeley National Laboratory (LBNL).

2.1 Simulation Setup

Figure 1 shows the simulated network topology. Concerning to simulation setup, the simulated network consists of one centralized router and three nodes. Each node connects to the router by a full-duplex link with 500 kbps of bandwidth and 50 ms of propagation delay. The queue limit is set to 100 packets.

All TCP connections transmit 100 segments. Each segment has 536 byte of payload and 40 bytes of header. Delayed ACK is enabled, and the maximum window size is 100 segments. To perform simulations in several network congestion levels, we changes the average number of TCP connections generated in 1 secnd. The average number is varied ranging from 1 connection to 3.5 connections with every 0.25 connections. We do not consider any traffic other than the TCP connections.

Additionally, we apply no error module in our simulations. In reality, segment losses occur caused by bit error on a transmission link. However, we would like to investigate a pure congestion control mechanism of TCP. So, we do not handle the influence of non-congestion related losses in this paper. In other words, all segment losses in our simulations are derived by queue overflow at forwarding nodes.

The source and the destination nodes are selected randomly based on exponential distributions. We set simulation time to 3,000 second.

In this simulation, we investigate and compare three variants of TCP-Reno, NewReno[8], and Sack[14]. Below, we shortly describe their differences. All the current implementations of TCP are based on TCP-Tahoe that incorporated algorithms for slow-start, congestion avoidance, fast retransmit, and so on. TCP-Reno is basically similar to TCP-Tahoe but with a fast retransmit modification including fast recovery. After fast retransmit, unlike TCP-Tahoe which performs slow-start, TCP-Reno inflates its congestion window more rapidly by setting it to minmum (receiver window, slow start threshold + number of duplicate ACKs).

TCP-NewReno is a modification of TCP-Reno's fast recovery, which stays in fast recovery until all segment losses in window are recovered. TCP-NewReno uses information contained in partial ACK which is an ACK that acknowledges some but not all of the unacknowledged segments in the sender's window. In TCP-Reno, a partial ACK takes the sender out of fast recovery. On the other hand, in TCP-NewReno, a partial ACK received during fast recovery is taken as an indication that the segment following the partial ACK was lost and should be retransmitted. Thus, partial ACKs ensure that the lost segments are retransmitted without waiting for timeout.

TCP-SACK adds an additional capability that allows faster recovery in the presence of multiple segment losses. TCP-SACK provides information about out-of-order segments received by receiver. It can recover multiple segment losses per RTT.

2.2 Simulation Results

Figure 2 shows the number of finished connections within 3,000 seconds of simulated time. The vertical axis indicates the number of finished connections and the horizontal axis indicates the flow density that is the average number of connections generated in one second.

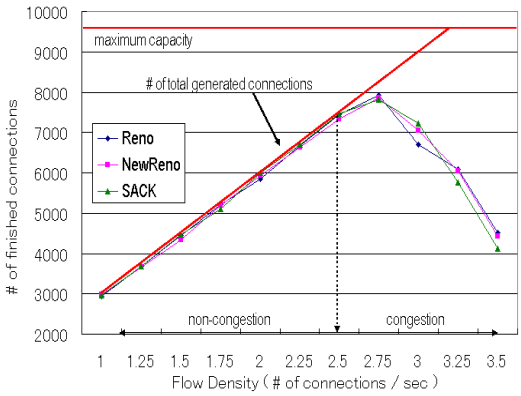

Fig. 2 The number of finished connections within simulated time.

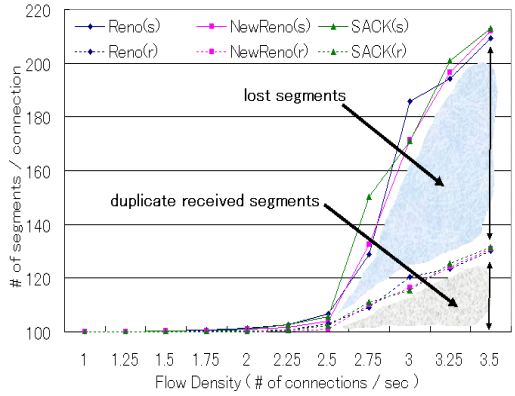

Fig. 3 The average number of sent and received segments per one connection.

Comparing the number of finished connections to the number of total generated connections, we can see the number of finished connections becomes decreasing at 2.5 of flow density. Since segment losses cause retransmission, some connections were backed off and did not finish the delivery of all segments within simulated time. The point, where experimental line separates from the logical line, can be referred as a *phase transition point* between non-congestion and congestion phases [22]. Phase transition behavior can be observed in various types of information traffics both in simulations and in real traffics [9),21)]. It is known that a numerical simulation based on a simple network topology can produce phase transition behaviors similar to those observed in real systems with complicated network topology. Namely, the phase transition phenomenon is quite universal as for information traffics. Actually, from this perspective, there are not remarkable differences between three variants of TCP-Reno, NewReno, and SACK.

To investigate the phenomenon of phase transition, we focus on a detailed behavior of each connection. **Figure 3** shows the average numbers of sent and received segments per one connection. Note that the vertical line starts from 100 segments. In ideal cases, the numbers of sent and received segments should be equal to 100 segments without any segment losses. It is acceptable that the number of sent segments increase because of retransmission for lost segments, as the mean flow density increases.

However, it is surprising that the number of received segments also increases as the mean flow density increases. Since each connection sends just 100 segments, the number of received segments should also be equal to 100 if there is no overlap. In other words, TCP receiver wastefully receives duplicate segments from TCP sender especially when the network is highly congested.

In the next section we will clarify the reason why TCP recevier receives duplicate segments from the sender.

3. The Principle of TCP

TCP guarantees reliable in-order delivery of data sent from the source to the destination. The TCP data unit exchanged between the source and the destination is called a segment. There are two kinds of segments, data segment and ACK segment. Using these two segments, TCP guarantees reliable data delivery by the following phases.

(1) A TCP sender transmits a data segment to a TCP receiver. (2) The TCP receiver replies an ACK segment to the TCP sender when TCP receiver receives the data segment. (3) The TCP sender confirms that the data segment was reached to the TCP receiver by receiving the ACK segment. As above, an ACK segment plays the role of the confirmation of data delivery. Then absence of an ACK segment causes a retransmission of a data segment. TCP does not care whether a data segment was lost in phase (1) or an ACK segment lost in phase (2). In both situations of a data segment loss and an ACK segment loss, TCP retransmits the data segment.

Generally in the Internet, a router does not care about whether an IP packet contains a data segment or an ACK segment. V. Paxson

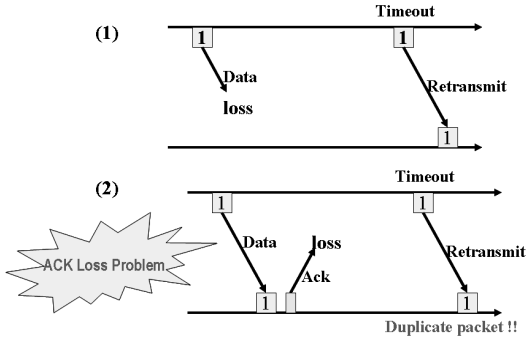

Fig. 4 Two types of retransmission timeout: (1) data segment loss, (2) ack segment loss.

reported [19] that the loss rate of ACK segments nearly equal to the loss rate of a data segments in actual network. Namely, ACK segment loss is not an unusual phenomenon.

TCP uses a cumulative acknowledgement of the expected next sequence number, and a later ACK segment contains all the information contained by earlier ACK segments. Under non-congested networks where TCP sender window inflates enough to generate multiple ACK segments by TCP receiver, later ACK segments can recover earlier ACK segments loss.

However, under congested networks, TCP sender window deflates due to frequent segment losses. As a result, the number of ACK segments generated by TCP receiver is few. Then it is highly possible that an ACK segment loss causes a retransmission of a data segment although the data segment successfully delivered to TCP receiver. In this way, duplicate data segments appear at TCP receiver.

Figure 4 shows two types of retransmission timeout. The first type is the situation where a data segment is lost, then TCP sender retransmits the data segment. The second type is the situation where an ACK segment is lost, then the TCP sender retransmits the data segment. In this situation, two identical data segments arrive at the TCP receiver. Since the duplicate data segment is discarded at TCP receiver, it can be said that the retransmission is wasteful. This phenomenon occurs when congestion level exceeds over a phase transition point which was described in the previous section.

4. Introduction of LEAP

According to the results of previous simulations, we found that the TCP receiver receives unneeded duplicate segments in the congested phase. Obviously, it is caused by ACK segment loss described in Section 3. To resolve this problem, we propose a TCP modification called LEAP. The reason why we named the scheme LEAP is that our scheme leaps a data segment of conventional TCP retransmission and transmits a new data segment when an ACK segment loss is suspected. LEAP has the following features:

- **Suspecting ACK loss**
  The key idea underlying LEAP is to suspect ACK segment losses, and then TCP sender adjusts TCP sender's behavior not to send a duplicate data segment to TCP receiver. When the retransmission timer for a segment expires, LEAP estimates whether the retransmission is actuated by a data segment loss or an ACK segment loss. Receiving duplicate ACK segments helps LEAP to expect that the possibility of an ACK segment loss is low. This is because duplicate ACK segment is generated by a data segment loss or a data segment disorder.
- **Sending LEAP segment**
  When LEAP suspects that the retransmission is generated by an ACK segment loss, LEAP sends a new sequenced data segment which outranges the current TCP window. The new segment is called the LEAP segment. Note that TCP does not inflate own window in order to send the LEAP segment. This is done by separating process from TCP sliding window.

LEAP Algorithm

Figure 5 illustrates the mechanism of (1) LEAP by comparing with (2) TCP-Reno. In both figures, TCP sender with 4 segments of window size transmits 4 data segments without waiting an ACK segment from TCP receiver. In this scenario, we assume that No.4 data segment and all ACK segments are lost due to congestion at an intermediate node.

In TCP-Reno when retransmission timer for No.1 data segment expires, TCP sender retransmits No.1 segment. It results receiving a duplicate data segment at TCP receiver. On the other hand, in LEAP, when retransmission timer for No.1 data segment expires, LEAP transmits No.5 data segment instead of transmitting No.1 data segment.

Just after receiving ACK segment, in both cases, TCP sender goes into slow start phase with increasing congestion window by 1 seg-

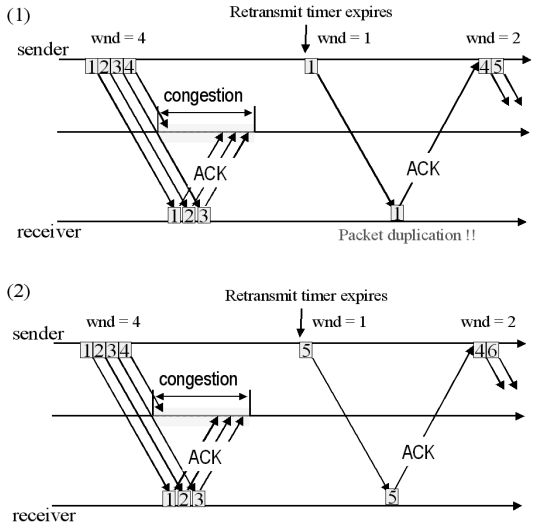

Fig. 5 The mechanism of LEAP by comparing with TCP-Reno. (1) TCP-Reno, (2) LEAP.

ment size. In this phase, LEAP will skip No.5 data segment because the data segment has been already transmitted and acknowledged. Consequently, TCP sender transmits No.4 and No.6 segments.

The suspect of an ACK segment loss by LEAP is not accurate. This means LEAP suspects that an ACK segment is lost while a data segment is lost. Here, we describe what will happen and how LEAP will behave when the LEAP suspect is incorrect.

To explain this process, we use Fig. 5 again. In this situation, actually, the retransmission of No.5 data segment of LEAP is undesirable. This is because the ACK segment for No.5 data segment requires No.4 data segment. In conventional TCP mechanism, an ACK requiring a segment of lower sequence number than transmitted segment cannot inflate congestion window. As a result, the TCP sender is forced to wait for a timeout to retransmit No.4 data segment.

To avoid this overhead, LEAP applies a few modifications to the TCP slow start mechanism. Until a sequence number of a received ACK segment becomes higher than that of a LEAP segment, a sender inflates window whenever it receives an ACK segment. By this revision LEAP can avoid waiting expiration of retransmission timer even though the suspect of ACK segment loss is undesirable.

5. Implementation of LEAP

In this section, we describe our implementation of LEAP in detail, using the FreeBSD TCP-Reno source code as our initial starting point. The LEAP algorithm is only added to retransmit timer and slow start parts of the TCP sender. We add two variables to TCB: *leap_seqno*, and *leap_flg*.

5.1 Two Variables Added to TCB

TCP maintains TCP specific information into TCP Control Block (TCB) structure. For implementation of LEAP, we add two variables, leap_seqno and leap_flg, to TCB.

- **leap_seqno variable**
  The leap_seqno variable maintains a sequence number of a leap segment that is transmitted instead of a timed out segment. The main purpose of setting this variable is to avoid sending the leap segment twice. Receiving an acknowledgement of the leap segment, TCP sender goes into slow start mode. If TCP window includes a segment that has the sequence number, which is kept in leap_seqno, TCP will send next sequenced segment instead of the segment.
- **leap_flg variable**
  The leap_flg variable is used to determine whether the leap segment is transmitting or not. When TCP sender transmits a leap segment, the leap_flg variable is set to 1. When TCP sender receives an ACK segment, the leap_flg variable is set to 0. If a retransmission timer expires when the leap_flg variable is 1, TCP sender retransmits the leap segment.
  There is no way to decide whether a received ACK is the one against last transmitted segment or not. Hence, setting leap_flg may mistake by a delayed ACK segment against previously transmitted segments. However, it can be said that is rare case because a value of retransmission timer is dynamically set based on round trip time and is big enough compared to round trip time.

5.2 Consideration on Retransmission Timer

A current TCP implementation aborts a connection when a sender retransmits an identical segment twelve times without receiving its acknowledgements. An exponential backoff is applied to the retransmission timeout (RTO). This is done by multiplying the RTO by a value

from the array defined in TCP source code.

In LEAP a first transmission of a leap segment is regarded as a second transmission of a timed out segment. To keep compatibility to a current TCP implementation, the RTO of the leap segment is calculated as a second transmission. When the retransmission timer for a leap segment expires, the leap segment is retransmitted since TCP sender knows that the leap segment does not reach to the TCP receiver. If the number of leap segment's retransmission equals to twelve, TCP aborts the connection.

5.3 Consideration on Receiver's Buffer Size

TCP window is set to *min*(*awnd, cwnd*), where *awnd* is the receiver's advertised window, and *cwnd* is the sender's congestion window. In heavily congested networks, TCP segments might be lost due to congestion, leading to frequent retransmissions. In this situation the value of *cwnd* is usually smaller than the value of *awnd*. ACK segment losses, however, arises not only in congested networks but also in non-congested networks. Even though TCP sender sends a segment with exceeding the value of *cwnd*, it can be successful in delivering the segment. However, when TCP sender sends a segment with exceeding the value of *awnd*, TCP does not succeed in delivering the segment. This is because TCP receiver has no buffer capacity to keep the received segment. In LEAP, if the sequence number of leap segment exceeds the value of *awnd*, LEAP does not transmit the leap segment. Instead of this, LEAP retransmits timed out segment, which is a normal TCP behavior.

5.4 Consideration on Segments with Special Control Flag

All the TCP segments are not valid when they are delivered at out-of-order. Some TCP segments with SYN, FIN, PUSH, or RST flags in their headers are used to control a TCP connection. To send a leap segment by leaping these segments is meaningless for the TCP connection. For example, if TCP sender transmits a leap segment with retransmission timeout for SYN segment, the leap segment would not be acceptable in TCP receiver. In LEAP, when retransmission timer for those segments expires, LEAP would not transmit a leap segment. Instead of this, the TCP sender transmits a timed out segment. It is easy to perform this process by checking flag bit in TCP header.

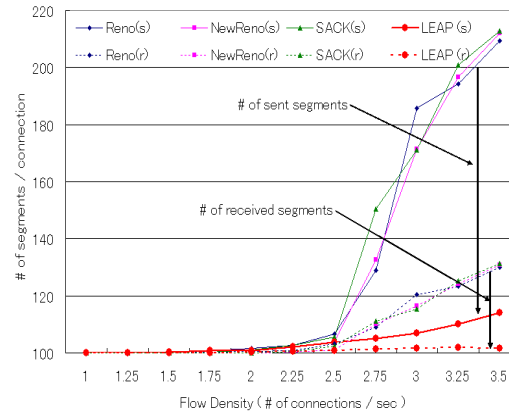

Fig. 6 Comparison of the numbers of sent segments and received segments per one connection.

6. Performance Evaluation

In this section we evaluate the LEAP performance by simulation. All simulations were performed by using the *ns-2*[16] simulator. To evaluate the performance of the LEAP, we compared LEAP with TCP-Reno, TCP-NewReno, and TCP-SACK. A simulated network topology and detailed configuration are the same with the simulations described in Section 2.

6.1 Comparison of the Number of Sent and Received Segments

We examined how many sent and received segments are generated by LEAP. **Figure 6** shows the average numbers of sent and received segments in one connection. In addition we represent **Fig. 7** by focusing on the number of duplicate received segment par one connection. We clearly found that LEAP reduces the number of unneeded duplicate received segments. Since the total number of segments in one connection is 100, most of unneeded duplicate received segments were eliminated by LEAP. There are two reasons why the number of this does not equal to 100. An ACK segment for special segments like SYN, FIN, PUSH, and RST can be lost, or an ACK segment for a leap segment can be lost. Then, the duplicate special segment or leap segment was received at TCP receiver. Additionally, we can find that LEAP reduces not only the number of received segments but also the number of sent segments. This is because the reduction of the number of unneeded duplicate received segments reduces the loss ratio in overall networks.

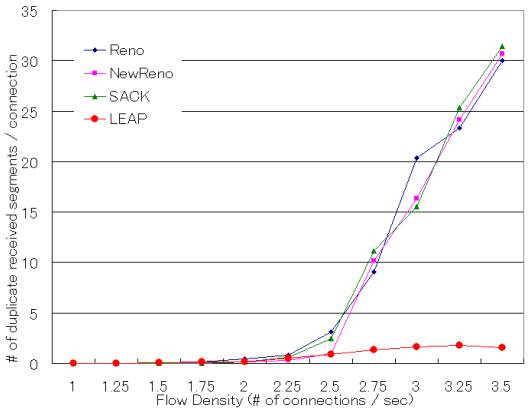

Fig. 7 Comparison of the numbers of duplicate received segments per one connection.

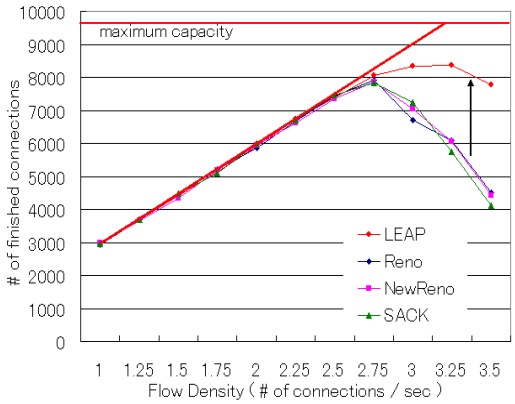

Fig. 8 Comparison of the number of finished connections within simulated time.

6.2 Comparison of the Efficiency in Network Utilization

Figure 8 shows the number of finished TCP connections within simulated time. **Table 1** shows the ratio of finished connections to generated connections. We find that the number of finished TCP connections tends to decrease from 2.75 flow density in Reno, NewReno, SACK. This is because phase transition occurs at approximately 2.75 flow density in Reno, NewReno, SACK based simulations. On the other hand, in LEAP, the number of finished TCP connections tends to decrease from 3.25 flow density. In the congested phase at 3.5 flow density, LEAP obviously shows higher performance than other TCP variants. It can be said that LEAP drastically improves the performance of network utilization by reducing wasted segments.

Table 1 Ratio of finished connections to generated connections (FD: Flow Density (conn/sec)).

FD	Reno	NewReno	SACK	LEAP
2.50	99.3%	97.9%	99.4%	98.5%
2.75	96.1%	95.3%	94.8%	97.8%
3.00	74.5%	78.3%	80.4%	92.7%
3.25	62.4%	62.1%	59.0%	85.8%
3.50	42.9%	42.2%	39.2%	74.0%

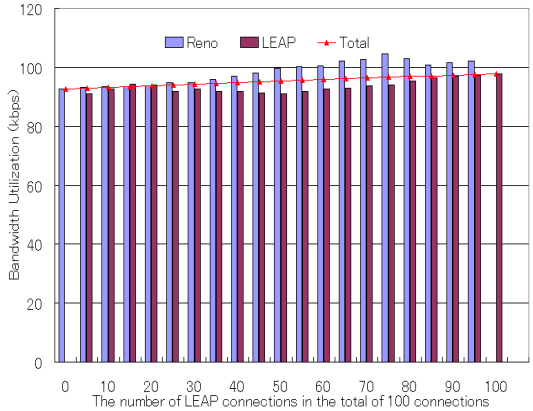

Fig. 9 The average throughput with changing a ratio of LEAP connections. The total number of connections is fixed to 100.

6.3 LEAP Influence on Coexisting TCP Flows

One of the important questions about proposing a new TCP modification is, does our proposed model disturb the traditional TCP model when both coexist in a network. In this section we investigate the LEAP influence on coexisting TCP flows. With simulations we use simple network topology where a single 10 Mbps link with 2 ms latency is shared by 100 TCP flows. We perform simulations with changing the ratio of the Reno and LEAP flows in 100 TCP flows. All flows are generated at one node at random time based on exponential distributions.

We calculated bandwidth utilization of each flow within simulation time (3,000 seconds). Each source sends data segments infinitely. **Figure 9** shows the simulation results. Each data point was plotted by averaging 100 realizations. This figure implies two important points. One is the average throughput of Reno is not degraded with increasing the ratio of LEAP connections. Shortly, it can be said that LEAP does not disturb Reno. Second is that the average throughput of the total connection improves as the ratio of LEAP connections becomes larger. We performed the same simula-

tions by comparing LEAP with NewReno and SACK. The same characteristics has been confirmed in the simulation results.

7. Related Works and Discussion

This section describes previous works on TCP performance improvements by focusing ACK behaviors. Some issues for LEAP performance are also discussed.

7.1 TCP Optimization with Focusing ACK Behaviors

For TCP an ACK segment is not only a confirmation of data delivery but also an important signal for detecting network congestion. TCP flow control is called self-clocking that allows automatic adjustment of the transmission speed to the bandwidth and propagation delay of the path by using ACK information. Since a mechanism for generating ACK segment effects to the performance of TCP, some researches [11] have been focusing on ACK behaviors. M. Allman pointed out the delayed acknowledgement mechanism hurts TCP performance, especially during slow start. He proposed and simulated three modifications [2] for TCP acknowledgement mechanism. However, this research is different from ours in the sense that they aim to avoid negative segment transfer in TCP startup phase. In LEAP we are seeking to eliminate duplicated data segments caused by an ACK segment loss.

7.2 Does LEAP Violate the Concept of the TCP Window ?

A leap segment is transmitted independently with the TCP window. To keep compatibility with conventional TCP, LEAP should not violate the concept of the TCP window. The TCP window has two key features; the first is for congestion control and the other is for synchronization between sender and receiver. Concerning congestion control, TCP transmits a leap segment instead of retransmission for a timed out segment. Therefore, the number of total transmitting segments is the same in both LEAP and Reno. On the other hand concerning synchronization between sender and receiver, a leap segment is transmitted only when sequence number of the leap segment is within receiver's advertised window. These careful implementations can keep compatibility between LEAP and the conventional TCP.

8. Summary

In this paper, we firstly investigated numerically the performance of TCP dynamics in congested networks where network resources are shared by many flows. We found that the losses of ACK segments at receivers degrade the performance of network resource utilization due to resulting production of duplicated data segments. This phenomenon can be seen when network congestion level exceeds a certain critical point. We referred to this point as the *phase transition point*, as it is the border point between the non-congestion and congestion phases. Such phase transition phenomenon can be observed quite universally in information traffics.

Unneeded duplicate segments not only waste network resource but also increase the overall network loss rate. Based on these simulation results we proposed a new scheme LEAP, for improving TCP performance of congested networks. LEAP performs based on the smart judgment, which estimates whether a retransmission of a data segment is invoked by a data segment loss or an ACK segment loss. In heavily congested networks having multiple connections, we showed that LEAP eliminates nearly all of duplicated segments without disturbing coexisting TCP flows. Additionally, LEAP can be applied with minor revision of TCP and it eases overall network congestion level considerably by the effect of elimination of unneeded duplicated data segments. Our simulation results showed that LEAP improves overall network utilization more than 120% higher than conventional TCP variants under congested networks.

References

1) Ahn, J.S., Danzig, P.B., Liu, Z. and Yan, L.: Evaluation of TCP Vegas: Emulation and Experiment, *Proc. ACM SIGCOMM'95* (Oct. 1995)
2) Allman, M.: On the Generation and Use of TCP Acknowledgments, *ACM Computer Communications Review*, Vol.28, No.5, (Oct. 1998)
3) Balakrishnan, H., et al.: TCP Behavior of a Busy Internet Server: Analysis and Improvements, *Proc. IEEE INFOCOM'98* (March 1998)
4) Brakmo, L.S., O'Malley, S.W. and Peterson, L.L.: TCP Vegas: New Techniques for Congestion Detection and Avoidance, *Proc. ACM SIGCOMM'94* (May 1994)
5) Casetti, C., Gerla, M., Mascolo, S., Sanadidi, M.Y. and Wang, R.: TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links, *Proc. ACM/IEEE MOBICOM*

2001 (July 2001)
6) Floyd, S. and Fall, K.: Promoting the Use of End-to-End Congestion Control in the Internet, *IEEE/ACM Transactions on Networking*, Vol.7, No.4 (Aug. 1999)
7) Floyd, S., Handley, M., Padhye, J. and Widmer, J.: Equation-Based Congestion Control for Unicast Applications, *Proc. ACM SIGCOMM 2000* (Aug. 2000)
8) Floyd, S. and Henderson, T.: The NewReno Modification to TCP's Fast Recovery, Request For Comments: 2582 (Apr. 1999)
9) Fukuda, K., Takayasu, H. and Takayasu, M.: Observation of Phase Transition Behaviors in the Internet Traffic, *Advances in Performance Analysis*, No.2, pp.45–66 (1999)
10) Hoe, J.C.: Improving the Start-up Behavior of a Congestion Control Scheme for TCP, *Proc. ACM SIGCOMM'96* (Aug. 1996)
11) Lakshman, T.V., Madhow, U. and Suter, B.: Window-based Error Recovery and Flow Control with a Slow Acknowledgement Channel: A Study of TCP/IP Performance, *Proc. IEEE INFOCOM'97* (Apr. 1997)
12) Mahdavi, J. and Floyd, S.: Tcp-friendly unicast rate-based flow control, *Technical note sent to the end2end-interest mailing list* (Jan. 1997)
13) Mathis, M. and Mahdavi, J.: Forward Acknowledgment: Refinding TCP Congestion Control, *Proc. ACM SIGCOMM'96* (Aug.1996)
14) Mathis, M., Mahdavi, J., Floyd, S. and Romanow, A.: TCP Selective Acknowledgment Options, Request For Comments: 2018 (Oct. 1996)
15) Mathis, M., Semske, K., Mahdavi, J. and Ott, T.: The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm, *Computer Communication Review*, Vol.27, No.3 (July 1997)
16) UCB/LBNL/VINT Network Simulator ns (version 2). http://www.isi.edu/nsnam/ns/
17) Padhye, J., Firoiu, V., Towsley, D. and Kurose, J.: Modeling TCP Reno Performance: A Simple Model and its Empirical Validation, *IEEE/ACM Transactions on Networking*, Vol.8, No.2 (Apr. 2000)
18) Parsa, C. and Garcia-Luna-Aceves, J.J.: Improving TCP Congestion Control over Internets with Heterogeneous Transmission Media, *Proc. IEEE ICNP'99* (Nov. 1999)
19) Paxson, V.: End-to-End Internet Packet Dynamics, *Proc. ACM SIGCOMM'97* (Sept.1997)
20) Sinha, P., Venkitaraman, N., Sivakumar, R. and Bharghavan, V.: WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks, *Proc. ACM/IEEE MOBICOM'99* (Aug. 1999)
21) Takayasu, M., Takayasu, H. and Fukuda, K.: Dynamic Phase Transition Observed in the Internet Traffic Flow, *Physica*, A, No.277, pp.824–834 (Sept. 2000)
22) Takayasu, M., Takayasu, H. and Sato, T.: Critical Behavior and 1/f Noise in Information Traffic, *Physica*, A, No.233 (1996)
23) Tamura, Y., Tobe, Y. and Tokuda, H.: EFR: A Retransmit Scheme for TCP in Wireless LANs, *Proc. IEEE LCN'98* (Oct. 1998)

(Received August 25, 2003)
(Accepted March 5, 2004)

Yosuke Tamura received the B.S. degree in environmental information, and the M.S. and Ph.D. degrees in Media and Governance from Keio University in 1997, 1999, and 2001, respectively. In 2002, he joined Sony Computer Science Laboratories, Inc. His research focuses on network architectures for mobile and sensor networks. He is a member of ACM, IEEE, and the IPSJ.

Misako Takayasu received the Ph.D. degree in Physics from Nagoya University in 1987. She has been an Associate Professor at Graduate School of Computational Intelligence and Systems Science, Tokyo Institute of Technology since 2004. Her research focuses on Statistical Physics and Information Physics. She is a member of Physical Society of Japan.

Hideki Takayasu received the Ph.D. degree in Physics from Nagoya University in 1980. He joined Sony Computer Science Laboratories, Inc. as a senior researcher in 1997. His research focuses on Fractal theory and Econophysics. He is a member of Physical Society of Japan.