Master's Thesis

Title

Achieving Scalability and Self-Adaptivity to Network Bandwidth and Delay for Measurement-based TCP Congestion Control

Supervisor

Professor Masayuki Murata

Author

Tomohito Iguchi

February 15th, 2005

Department of Information Networking

Graduate School of Information Science and Technology

Osaka University

Master's Thesis

Achieving Scalability and Self-Adaptivity to Network Bandwidth and Delay for Measurement-based TCP Congestion Control

Tomohito Iguchi

Abstract

The heterogeneity and complexity of the Internet are increasing with the rapid development of such networking technology as wireless and DSL/FTTH access network links, gigabit/terabit-level high-speed backbone network links, and the explosive growth of the Internet population. As network link bandwidth increases, server and client machines directly connect to higher-speed networks to deliver gigabyte/terabyte data to other hosts. When TCP Reno, which is the de facto standard transport layer protocol in the current Internet, is used for such high-speed data transmission, it cannot achieve sufficient throughput because of the nature of its essential congestion control mechanism. The main reason is that TCP Reno does not utilize the bandwidth information of the network path, one of the most important factors for congestion control in the Internet. The variants of TCP Reno previously proposed for large bandwidth and long delay networks have the same shortcomings.

In this thesis, we propose a new congestion control mechanism based on the measurement of the network bandwidth information. We adopt inline network measurement since it can measure physical and available bandwidths from data/ACK packets transmitted by active TCP connections in an inline fashion. Our proposed mechanism directly obtains bandwidth information by using the inline network measurement technique and adjusts congestion window size by using an algorithm based on a logistic growth model and a Lotka-Volterra competition model from biophysics.

Through mathmatical analysis, we compare the proposed mechanism with traditional TCP Reno and its variants, and exhibit its effectiveness for scalability to the network bandwidth and delay, convergence time, fairness among competing connections, and stability. Furthermore, we confirm that the proposed mechanism has the above characteristics through extensive simulation experiments.

Keywords

Transmission Control Protocol (TCP), Congestion Control, Inline Network Measurement, Physical Bandwidth, Available Bandwidth, Logistic Growth Model, Lotka-Volterra Competition Model

Contents

List of Figures

1 Introduction

Transmission Control Protocol (TCP) [1] is the de facto standard transport layer protocol of the current Internet first designed in the 1970s: the first Request for Comments (RFC) on TCP was released in 1981 [2]. TCP has also been frequently modified and enhanced to accommodate the development of the Internet [3-5].

TCP has various functions to realize reliable and efficient data transmission on the network. The congestion control mechanism [1] is one of the most important. Its main purpose is to avoid and resolve network congestion and to distribute network bandwidth fairly among competing connections. To do that, TCP employs a window-based congestion control mechanism that adjusts data transmission speed by changing congestion window size. A congestion window indicates the maximum amount of data that can be sent out on a connection without being acknowledged.

TCP Reno is the most popular version of TCP in current operating systems. Its window size control algorithm allows a TCP sender to continue to additively increase its congestion window size until it detects a packet loss (or losses), decreasing it multiplicatively when a packet loss occurs. This is called an Additive Increase Multiplicative Decrease (AIMD) policy. In [6], the authors argue that an AIMD policy is suitable for efficient and fair bandwidth usage in a distributed environment, if congestion indication signals are simultaneously distributed to all connections.

However, with increases in the heterogeneity and the complexity of the Internet, many problems have emerged in TCP Reno's congestion control mechanism ([7-11] for example). The main reasons are that the congestion signals are only indicated by packet loss and TCP Reno uses fixed AIMD parameter values in increasing and decreasing window size, whereas they should be changed according to the network environment.

For example, a previous paper [10] maintained that the throughput of TCP connections decreases when it traverses wireless links, since TCP cannot distinguish congestion-oriented packet loss caused by network congestion, and wireless-oriented packet loss caused by link loss and/or handoff. Some solutions can be found in [12-14]. Another problem is the low throughput of TCP connections in large bandwidth and long delay networks. In [11], the authors argued that a TCP Reno connection cannot fully utilize the link bandwidth of such networks, since the increasing parameter, which is one packet per a Round Trip Time (RTT), is too small and the decreasing parameter, which halves the window size when a packet loss occurs, is too large for networks

with a large bandwidth-delay product. Although there are many solutions to the problem [11, 15, 16], almost all of them inherit the fundamental congestion control mechanism of TCP Reno: the AIMD mechanism triggered by the detection of packet losses in the network. The mechanisms improve the throughput by adjusting the increasing and decreasing parameters statically and/or dynamically. However, most previous papers focused on changing the AIMD parameters for accommodating particular network environments. That is, since those methods may employ ad hoc modifications for a certain network situation, their performance is not clear when applied to other network environments. Therefore, we believe they have not essentially solved the problem.

Because window size indicates the maximum amount of packets that TCP can transmit for one Round Trip Time (RTT), adequate window size for a TCP connection is equal to the product of the available bandwidth and the propagation delay between the sender and receiver hosts. TCP Reno measures the RTTs of the network path between sender and receiver hosts by checking the departure times of the data packets and the arrival times of the corresponding ACK packets. However, it does not have an effective mechanism to recognize the available bandwidth. This is the essential explanation of the problem: it cannot adjust window size to an adequate value under various network environments. In a sense, the traditional TCP Reno can be considered a tool that measures available bandwidth because of its ability to adjust the congestion window size to achieve a transmission rate appropriate to the available bandwidth. However, it is ineffective since it only increases window size until a packet loss occurs. In other words, it induces packet losses to obtain information about the available bandwidth(-delay product) of the network. That is, even when the congestion control mechanism of TCP works completely, the TCP sender experiences packet losses in the network at some intervals. Since all modified versions of TCP using AIMD policy contain this essential problem, they cannot avoid periodic packet losses. We believe, therefore, if a TCP sender recognizes the bandwidth information quickly and adequately, we can create a better mechanism for congestion control in TCP.

Fortunately, many measurement tools have been proposed in the literature [17-22] that measure the physical and available bandwidths of network paths. However, we cannot directly employ those existing methods into TCP mechanisms since they utilize a lot of test probe packets; they also require too much time to obtain one measurement result. We have proposed a method called Inline measurement TCP (ImTCP) that avoids these problems in [23-25]. It does not inject extra traffic into the network, and instead it estimates the physical/available bandwidths of the network

path from data/ACK packets transmitted by an active TCP connection in an inline fashion. Furthermore, since the ImTCP sender obtains bandwidth information every 1–4 RTTs, it can follow the traffic fluctuation of the underlying IP network well. Furthermore, because it is implemented at the bottom of the TCP layer, all kinds of TCP's congestion control mechanisms can include this measurement mechanism.

In this thesis, we propose a new congestion control mechanism of TCP that utilizes the information of physical and available bandwidths obtained from an inline measurement technique. The proposed mechanism does not use ad hoc algorithms such as TCP Vegas [26]: instead it employs algorithms that have a mathematical background that enable us to mathematically discuss and guarantee their behavior even though posing a simplification of the target system. More importantly, it becomes possible to give a reasonable background of our control parameter selections within TCP, instead of conducting intensive computer simulations and/or choosing parameters in an ad hoc fashion. We designed a window size control algorithm that aims to quickly adjust window size to an adequate value by using bandwidth information to fairly distribute bandwidth among competing connections. Therefore, we borrowed algorithms from biophysics: the logistic growth model and the Lotka-Volterra competition model [27], which describe changes in the population of species, are applied to the window updating algorithm of the proposed TCP. This application can be done by considering the population of a species as a window size of a TCP connection, the carrying capacity of the environment as physical bandwidth, and interspecific competition among species as bandwidth share among competing TCP connections. We present in detail how to apply the logistic growth and Lotka-Volterra competition models to the congestion control algorithm of our TCP as well as analytic investigations of the proposed algorithm. Then, we can utilize the existing discussions and results on various characteristics of the mathematical models, including scalability, convergence, fairness and stability. Giving those characteristics to TCP is the main objective of this thesis. We also present extensive simulation results to evaluate the proposed mechanism and show that, compared with traditional TCP Reno and other TCP variants, it utilizes network bandwidth effectively, quickly, and fairly.

The rest of this thesis is organized as follows. In Section 2, we introduce mathematical models in biophysics and inline network measurement we utilize in our proposed mechanism. In Section 3, we design the proposed TCP mechanism. In Section 4, we analyze and discuss the characteristics of the proposed mechanism. In Section 5, we present its effectiveness and performance through

various simulation experiments. We finally conclude this thesis and offer future work in Section 6.

2 Research Background

In this section, we briefly introduce the mathematical models in biophysics that are applied to the proposed mechanism in the next section, and the inline network measurement mechanism.

2.1 Mathematical Models

2.1.1 Logistic Model

The logistic equation is a formula that represents the evolution of the population of a single species over time. Generally, the per capita birth rate of a species increases as the population of the species becomes larger. However, since there are various restrictions on living environments, the carrying capacity of the environment exists, which is usually determined by the available sustaining resources. The logistic equation describes such changes in the population of the species as follows [27]:

$$\frac{d}{dt}N = \epsilon \left(1 - \frac{N}{K}\right) N \tag{1}$$

where t is time, N is the population of the species, K is the carrying capacity of the environment, and ϵ is the intrinsic growth rate of the species ($0 < \epsilon$).

Figure 1 shows changes in the population of the species (N) as a function of time where K = 100 and ϵ changes to 1.0, 2.0, 4.0 and 8.0. Looking at the lines, we can observe the following characteristics of the logistic equation: when N is much smaller than K, the increasing speed of N becomes larger as N increases. On the other hand, when N becomes close to K, the increasing rate decreases, and N converges to K. As ϵ increases, convergence time becomes smaller.

2.1.2 Lotka-Volterra Competition Model

The Lotka-Volterra competition model is famous for the population growth of two or more species including interspecific competition among them. In the model, Equation (1) is modified to include the effects of interspecific competition as well as intraspecific competition. The basic two-species Lotka-Volterra competition model with each species N_1 and N_2 having logistic growth in the

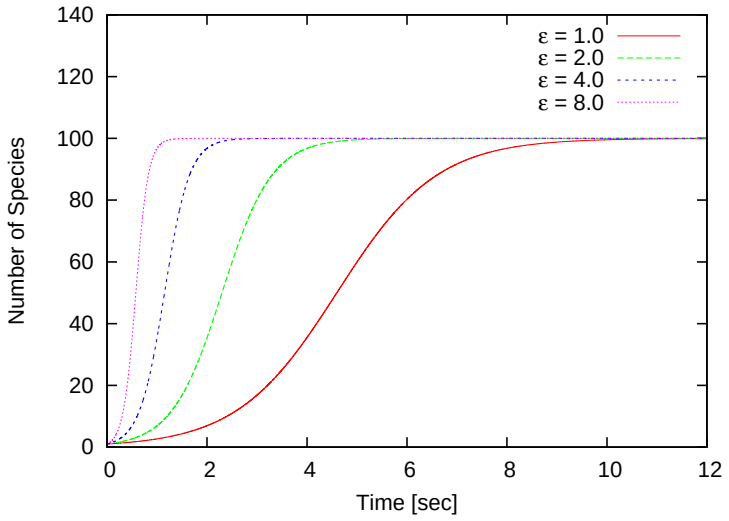


Figure 1: Changes in population of species with Logistic growth model

absence of the other is comprised of the following equations [27]:

$$\frac{d}{dt}N_1 = \epsilon_1 \left(1 - \frac{N_1 + \gamma_{12} \cdot N_2}{K_1}\right) N_1 \tag{2}$$

$$\frac{d}{dt}N_2 = \epsilon_2 \left(1 - \frac{N_2 + \gamma_{21} \cdot N_1}{K_2}\right) N_2 \tag{3}$$

where N_i, K_i, and ϵ_i are the population of the species, the carrying capacity of the environment, and the intrinsic growth rate of the species i, respectively. γ_{ij} is the ratio of the competition coefficient of species i on species j.

In this model, the population of species 1 and 2 does not always converge to a value larger than 0, and in some cases one of them becomes extinct. It depends on the value of γ_{12} and γ_{21}. It is a common characteristic that when the following conditions are satisfied, two species can survive in the environment [27]:

$$\gamma_{12} < \frac{K_1}{K_2}, \quad \gamma_{21} < \frac{K_2}{K_1} \tag{4}$$

Assuming that the two species have the same characteristics, they have the same values $K = K_1 = K_2$, $\epsilon = \epsilon_1 = \epsilon_2$, and $\gamma = \gamma_1 = \gamma_2$. Then, Equations (2) and (3) can be written as follows:

$$\frac{d}{dt}N_1 = \epsilon \left(1 - \frac{N_1 + \gamma \cdot N_2}{K}\right) N_1 \tag{5}$$

$$\frac{d}{dt}N_2 = \epsilon \left(1 - \frac{N_2 + \gamma \cdot N_1}{K}\right) N_2 \tag{6}$$

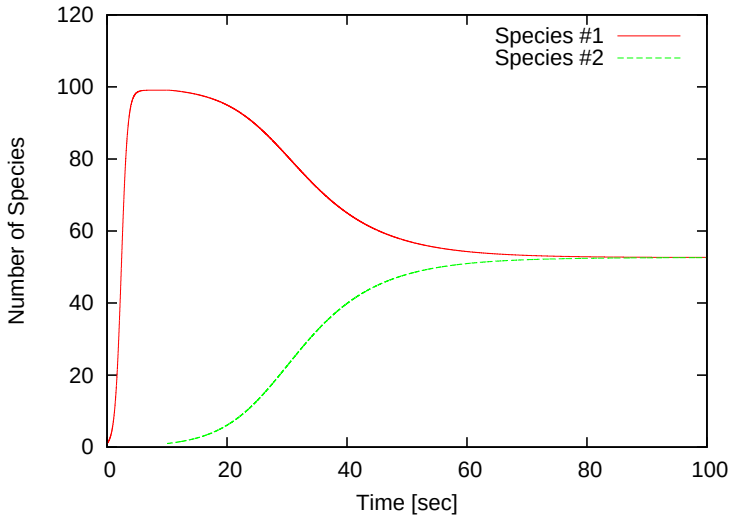


Figure 2: Changes in population of only two species with Lotka-Volterra competition model

Besides, Equation (4) can be written as $\gamma < 1$. Figure 2 shows the population changes in the two species by using Equations (5) and (6), where K = 100, ϵ = 1.95 and γ = 0.90, and species 2 join the environment 10 seconds after species 1. We can observe from this figure that the population of the two species converges quickly at the same value.

We can easily extend Equations (5) and (6) for n species as follows:

$$\frac{d}{dt}N_i = \epsilon \left(1 - \frac{N_i + \gamma \cdot \sum_{j=1, i \neq j}^{n} N_j}{K}\right) N_i \tag{7}$$

Figure 3 shows population changes in the ten species by using Equation (7), where K = 100, ϵ = 1.95 and γ = 0.90, and new species join the environment one after another. We can observe that ten species converge as two species do in Figure 2. Note that survival and convergence conditions are identical: that is, $\gamma < 1$. Even when two or more species exist, each independently utilizes Equation (7) for obtaining N_i, and the population of the species can converge to the value equally shared among competing species. Of course, this model can be directly applied to the congestion control algorithm of TCP because it has to obtain N_j. We discuss it in Subsection 3.1.

2.2 Inline Network Measurement

In [23-25], the authors proposed ImTCP, which is an inline network measurement technique for the physical and available bandwidths of network paths between TCP sender and receiver hosts. It can

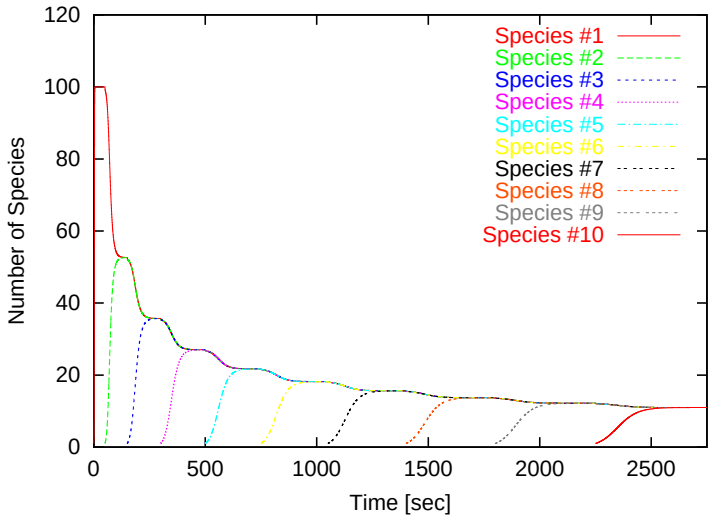


Figure 3: Changes in population of 10 species with Lotka-Volterra competition model

continuously measure bandwidths by using data and ACK packets of a TCP connection under data transmission. That is, the TCP sender transmits data packets at intervals determined by an inline measurement algorithm and checks the arrival interval times of the corresponding ACK packets to estimate bandwidth. Since it performs the measurement without transmitting additional probe packets into the network, it negligibly affects other network traffic. It can also quickly update the latest changes in bandwidths by frequently performing measurements (one result per 1–4 RTTs) as long as TCP transmits data packets. The authors have also proposed an implementation design of ImTCP, where the measurement program is located at the bottom of the TCP layer. It maintains the transmission/arrival intervals of TCP data/ACK packets by introducing a FIFO buffer between the TCP and IP layers. Note that the measurement algorithm does not affect TCP's congestion control algorithm, but only refers to the window size of the TCP sender. This means that the measurement algorithm can be applied to any TCP variants having a different congestion control algorithm.

3 Proposed Mechanism Design

3.1 Application to Window Size Control Algorithm

We consider that the changing population trends of species depicted in Figures 2 and 3 are ideal to control the transmission speed of TCP. That is, by using Equation (7) for the congestion control algorithm of TCP, rapid and stable link utilization can be realized.

To convert Equation (7) to a transmission rate control algorithm, we consider N_i as the transmission rate of a TCP sender i and K as the physical bandwidth of the bottleneck link. Furthermore, when applying Equation (7) to the congestion control algorithm for connection i, it is necessary for connection i to know the data transmission rates of all other connections that share the same bottleneck link. This assumption is quite unrealistic in the current Internet. Therefore, we approximate the sum of the data transmission rates of all of other connections by using the physical and available bandwidths as follows:

$$\sum_{j=1,i\neq j}^{n} N_j = K - A_i$$

Thus, Equation (7) becomes:

$$\frac{d}{dt}N_i = \epsilon \left(1 - \frac{N_i + \gamma \cdot (K - A_i)}{K}\right) N_i \tag{8}$$

where A_i is the available bandwidth for connections i. We assume that all connections share the same bottleneck link K.

A TCP sender controls its data transmission rate by changing its window size. To retain the essential characteristics of TCP and decrease the implementation overhead, we employ window-based congestion control in our proposed TCP by converting Equation (8) to obtain an increasing algorithm of window size in TCP. The window size of connection i, w_i, is calculated from N_i, the transmission rate, by the following simple equation:

$$w_i = N_i \tau_i$$

where τ_i is the minimum value of the RTTs of connection i, which is assumed to equal the propagation delay without a queueing delay in the intermediate routers between sender and receiver hosts. Now Equation (8) can be rewritten as follows:

$$\frac{d}{dt}w_i = \epsilon \left(1 - \frac{w_i + \gamma(K - A_i)\tau_i}{K\tau_i}\right) w_i \tag{9}$$

Finally, we integrate Equation (9) as follows:

$$w_i(t) = \frac{w_i(0)e^{\epsilon t\left\{1-\gamma\left(1-\frac{A_i}{K}\right)\right\}}\left\{K-\gamma\left(K-A_i\right)\right\}\tau_i}{w_i(0)\left(e^{\epsilon t\left\{1-\gamma\left(1-\frac{A_i}{K}\right)\right\}}-1\right)+\left\{K-\gamma\left(K-A_i\right)\right\}\tau_i} \tag{10}$$

In Equation (10), when we set the initial value of the window size ($w_i(0)$) and the current time to 0 (t = 0), we can directly obtain window size $w_i(t)$ at any time t. We use the above equation for the control algorithm of the window size of TCP connections.

The proposed congestion control algorithm is based on the traditional TCP Reno, and we integrate the inline network measurement technique in ImTCP [23-25]. Equation (10) requires the measurements of the physical and available bandwidths of a network path. Therefore, we use the same algorithm as TCP Reno for the window updating algorithm until measurement results are obtained through inline network measurements. In cases of packet loss (or losses), window size is decreased in an identical way to TCP Reno in both cases of timeout and fast retransmit [1].

When bandwidth information is obtained, the congestion control algorithm adjusts its window size using Equation (10). That is, When j-th ACK packet is received at sender TCP, we use Equation (10) to obtain the new value of the congestion window size of the TCP connection by the following calculation: set $w_i(0)$ to the current window size and t to the time duration from the arrival time of the (j-1)-th ACK packet to that of the j-th ACK packet. In Figure 4, we show the pseudocode of the congestion control algorithm of our proposed mechanism.

3.2 Implementation Issues

Equation (10) has the e^x calculation. Generally, exponentiation cannot be operated in the system kernel because of the lack of library and processing overhead. Therefore, we give the Taylor polynomial of degree 4 around x = a for function e^x as follows,

$$e^x \sim e^a \sum_{k=0}^{4} \frac{1}{k!}(x-a)^k$$

where a is an integer part of x (ex. a = 0 when $0 \leq x < 1$). By preparing e^a on a memory table for the limited range of a, we can calculate e^x with minimal processing overhead. In determining

```
// on event of receiving one ack
onEventACK()
{
   IF BW_measured = 1 THEN  // Bandwidth is measured?
      w_0 = cwnd_  // set current window size
      t_j = now()
      t = t_j - t_j1  // time duration
      cwnd_ = getCurrentCwnd(w_0, t)  // use Equation (10)
      t_j1 = t_j
   ELSE
      cwnd_ = cwnd_ + getRenoCwnd()  // use TCP Reno algorithm
   ENDIF
}
```

Figure 4: Pseudocode of proposed mechanism algorithm

the maximum value of a in our proposed mechanism, we consider the following equation:

$$\begin{aligned} x &= \epsilon t \left\{ 1 - \gamma \left(1 - \frac{A_i}{K} \right) \right\} \\ &\leq \epsilon t \end{aligned}$$

That is, when we assume that the maximum RTT of TCP connection is 10 [sec], we can determine the maximum value of a to $\lfloor 10\ \epsilon \rfloor$.

4 Characteristics of Proposed Mechanism

In this section, we analyze the various characteristics of the proposed mechanism such as scalability, convergence, stability, and so on. This analysis illustrates that the proposed mechanism essentially solves TCP Reno's problems.

4.1 Scalability and Convergence Time

Assuming physical bandwidth K and available bandwidth A are constant, the window size converges to a certain value in the proposed mechanism. The converged window size, which is denoted as w^*, can be obtained by setting dw/dt = 0 in Equation (9):

$$w^* = \{(1-\gamma)K + \gamma A\}\tau \tag{11}$$

We obtain time T_{proposed}, which is required to increase window size from w_0 to ρw^* ($0 < \rho < 1$, $w_0 < \rho w^*$), by using Equation (10) as follows:

$$\begin{aligned} T_{\mathrm{proposed}} &= \frac{1}{\epsilon\left\{1-\gamma\left(1-\frac{A}{K}\right)\right\}} ln\left(\frac{\rho}{1-\rho}\frac{w^*-w_0}{w_0}\right) \\ &\leq \frac{1}{\epsilon(1-\gamma)} ln\left(\frac{\rho}{1-\rho}\frac{w^*-w_0}{w_0}\right) \end{aligned} \tag{12}$$

where $0 \leq A \leq K$ is satisfied. In the case of TCP Reno, we can easily calculate time T_{reno}, the time necessary to increase window size from w_0 to w^*, as follows:

$$T_{\mathrm{reno}} = (w^* - w_0)\bar{\tau} = [\{(1-\gamma)K + \gamma A\}\tau - w_0]\bar{\tau} \tag{13}$$

where $\bar{\tau}$ is the average value of the RTTs of the TCP connection. Here, we ignore the effect of delayed ACK option [1] and focus only on the congestion avoidance phase of TCP Reno. In the case of HighSpeed TCP (HSTCP) [11], which is denoted by T_{hstcp}, is given by:

$$T_{\mathrm{hstcp}} \geq \frac{w^* - w_0}{a_{\max}}\bar{\tau} = \frac{\{(1-\gamma)K + \gamma A\}\tau - w_0}{a_{\max}}\bar{\tau} \tag{14}$$

where $a_{\max}$ is a parameter of HSTCP that indicates the maximum amount of window size increased in one RTT (equivalent to $a(W)$ in [11]). In a Scalable TCP (STCP), we calculate time T_{stcp} as follows:

$$T_{\mathrm{stcp}} = \frac{1}{a}\left(ln\frac{w^*}{w_0}\right)\bar{\tau} = \frac{1}{a}\left(ln\frac{\{(1-\gamma)K + \gamma A\}\tau}{w_0}\right)\bar{\tau} \tag{15}$$

where a is a STCP parameter, which is the amount the window size increased in receiving 1 ACK packet. In [15], a = 0.01 [packet] is the default value.

From Equations (13) and (14), it is observed that the time it takes to increase its window size is proportional to physical bandwidth K and propagation delay τ. It illustrates that the time it takes to utilize the bandwidth-delay product of the network path fully is proportional to the bandwidth-delay product. HSTCP proposed in [11] was designed as a new congestion control mechanism to resolve the problem in TCP Reno for high-speed and long delay networks. However, since the window size control algorithm of HSTCP is essentially based on the AIMD policy, it suffers from bad scalability to the bandwidth-delay product. STCP has a window size control algorithm based on Multiplicative Increase Multiplicative Decrease (MIMD) policy and describes logarithmic increases of time against increases of link bandwidth, as shown in Equation (15). Hence, it has good scalability to bandwidth. However, Equation (15) clearly shows bad scalability to propagation delay. On the other hand, the proposed mechanism increases logarithmically time against increases of link bandwidth and propagation delay in Equation (12).

4.2 Parameter Settings

The congestion control algorithm of the proposed mechanism has two parameters, γ and ϵ. In this subsection, we discuss the effect of those parameters and show some guidelines to configure γ and ϵ.

4.2.1 γ Setting

γ indicates the degree of the influence of the other competing connections that share the same bottleneck link. To converge window size to a positive value despite the physical bandwidth K_i of each connection, it is necessary to satisfy condition $0 < \gamma < 1$. From Equations (11) and (12), furthermore, we need to consider the trade-off between convergence speed and the final amount of packets accumulated within the buffer at the bottleneck link. That is, although smaller γ leads to faster convergence speed, it increases the queue size of the bottleneck router buffer when the window size is converged. By using Equation (11) we can easily obtain the sum of the window

size of n TCP connections as follows:

$$\sum_{i=1}^{n} w_i = \frac{n}{1+(n-1)\gamma} K\tau \tag{16}$$

where we assume that physical bandwidth K and the delay τ of each connection are identical. From Equation (16) queue size Q at the bottleneck link is given by:

$$Q = \frac{(n-1)(1-\gamma)}{1+(n-1)\gamma} K\tau \tag{17}$$

This equation shows that Q increases as n becomes larger. However, as n goes to infinity, we can obtain the following equation:

$$\lim_{n\to\infty} Q = \frac{1-\gamma}{\gamma} K\tau \tag{18}$$

That is, there exists an upper bound of the queue size against an increase of the number of concurrent TCP connections. Therefore, if the bottleneck link has a large enough buffer, the proposed mechanism will induce no packet losses regardless of the number of TCP connections. TCP Reno, HSTCP, and STCP, on the other hand, increase their window size until they fully utilize the buffer at the bottleneck link. As a result, they cannot avoid periodic packet losses. In the case of FAST TCP, we can estimate the total window size of n connections that share the bottleneck link. Just before the FAST TCP connection converges its window size, it updates window size according to the following equation [16]:

$$w_i \leftarrow (1-p)w_i + p\left(\frac{\tau}{\bar{\tau}} w_i + \alpha\right) \tag{19}$$

where α is a constant to determine the increment degree of the window size and p ($0 < p < 1$) is a smoothing coefficient parameter. From Equation (19), we obtain the following equation on converged window size:

$$w_i = \frac{\alpha\bar{\tau}}{\bar{\tau}-\tau} \tag{20}$$

Since w_i for each connection is independently calculated, the sum of the converged window size of n TCP connections is then calculated as follows:

$$\sum_{i=1}^{n} w_i = n\frac{\alpha\bar{\tau}}{\bar{\tau}-\tau} \tag{21}$$

We observe from Equation (21) that the sum of the window size is proportional to the number of TCP connections. So, to avoid packet losses, it is necessary to prepare a bottleneck link buffer based on the number of connections. We therefore conclude that FAST TCP cannot provide scalability to the number of concurrent TCP connections in the network.

4.2.2 ϵ Setting

ϵ determines convergence speed, as shown in Equation (9) and Figure 1. Generally, when we convert Equation (1) into the discrete equation, the population of the species does not converge with $\epsilon \geq 2$ [27]. In contrast, the window size updating algorithm proposed in Subsection 3.1 converts Equation (10) into a discrete equation in such a way that does not cause oscillation. Therefore, in the proposed algorithm, there is no limitation in ϵ, which means that as ϵ becomes larger, the window size converges faster. However, a value of ϵ too large makes the TCP sender transmit many packets in bursty fashion that may degrade the network performance.

Furthermore, there is another issue to be considered to set ϵ. In the logistic growth model (Equation (1)), on which the proposed mechanism is based, per capita birth rate is determined according to the current population of the species. In the proposed mechanism, the increase degree of window size is determined by using the bandwidth information obtained by ImTCP's inline measurement. Since the measurement algorithm in ImTCP utilizes the arrival time of ACK packets corresponding to the data packets transmitted by the sender host, the obtained measurement results experience some delay. Here, we consider the logistic growth model with delayed feedback described in the following equation:

$$\frac{d}{dt}N(t) = \epsilon \left(1 - \frac{N(t-\tau_d)}{K}\right) N(t) \tag{22}$$

where τ_d is the delay of the feedback information. When the population of the species changes with Equation (22), the population does not converge to a certain value even in a continuous-time model, if the following condition is satisfied [27]:

$$\tau_d > \frac{\pi}{2\epsilon}$$

That is, in Equation (22), it is necessary to satisfy $\epsilon \leq \pi/2\tau_d$ to converge the population. A similar limitation of ϵ exists in the proposed mechanism. This means that with a delay of the bandwidth information, changing window size too drastically causes oscillation of the window size.

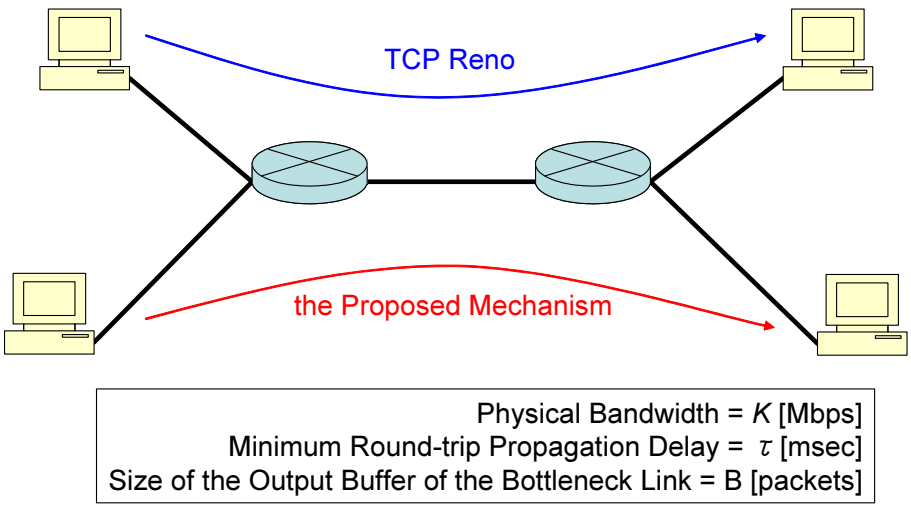


Figure 5: Model for fairness analysis

The delay in the proposed mechanism corresponds to the time it takes for the data (ACK) packets to traverse from the bottleneck link to the sender hosts. Since ImTCP needs 1–4 RTTs to measure the bandwidth information, we believe that the delay becomes about 1–2 RTTs. That is, the length of the delay depends on the RTT value of a TCP connection. In other words, by setting ϵ according to the RTT for each TCP connection, the proposed mechanism can avoid window size oscillation. However, using different ϵ brings different convergence speeds as shown in Figure 1, short-term unfairness among connections with different RTTs might happen. We conclude that we should take this trade-off into consideration when configuring ϵ in the proposed mechanism.

4.3 Competition with TCP Reno

In this subsection, we investigate the fairness property of the proposed mechanism against competing TCP Reno connections. For that objective, we compare the throughput of two TCP connections, which deploy TCP Reno and the proposed mechanism share a bottleneck link, by analyzing changes in congestion window sizes. Figure 5 depicts the network model for analysis, where K is the physical bandwidth, τ is the minimum round-trip propagation delay except the queueing delay, and B is the size of the output buffer adopting a TailDrop scheme, of the bottleneck link.

As explained above, the proposed mechanism converges its window size to a certain value while TCP Reno continues increasing its window size until a packet loss occurs. Hence, even

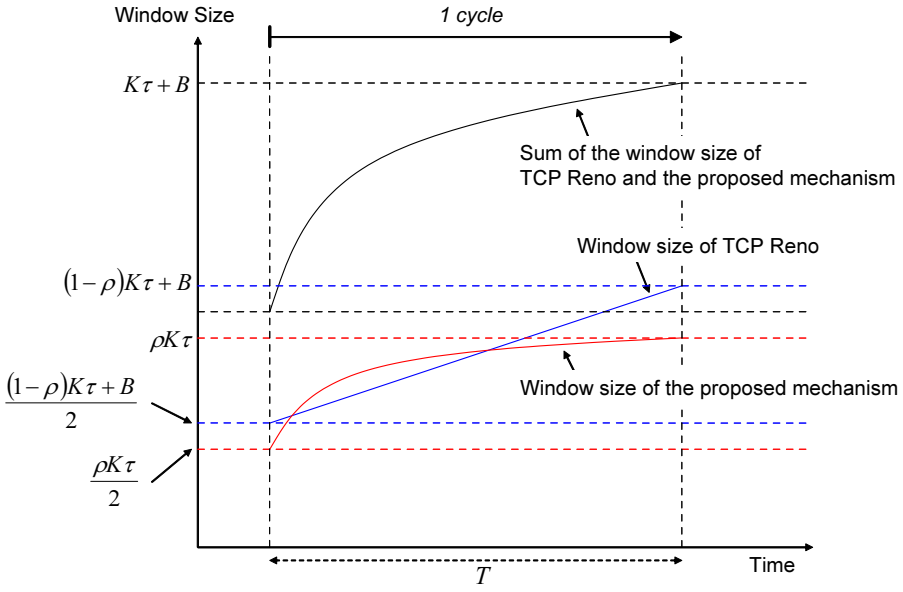


Figure 6: Changes in the window sizes of TCP Reno and the proposed mechanism

when both TCP connections compete at the bottleneck link bandwidth, periodic packet losses occur at the buffer. We, therefore, assume that both TCP connections experience a packet loss when the buffer becomes fully utilized. Therefore, the window size of the two TCP connections changes cyclically, triggered by packet losses. Figure 6 describes such changes in the window size. Here we define one cycle as a period between two packet losses and denote the length of the cycle as T. We assume that the received socket buffer of each TCP connection is large enough not to limit the congestion window size evolution.

The proposed mechanism obtains the physical and available bandwidths with an inline network measurement. In this analysis, we assume that the sender of the proposed mechanism can obtain precise physical bandwidth information. On the other hand, when several TCP and/or ImTCP connections share bottleneck link bandwidth, the ImTCP sender cannot obtain the available bandwidth as expected, and so the obtained value becomes almost equal to the available bandwidth for all TCP connections sharing the bottleneck link [24]. This is because a TCP connection tends to transmit bursty data packets [28], and then packets from each TCP connection traverses the bottleneck link in a back to back fashion. Therefore, we suppose in this analysis that the measured available bandwidth A is equivalent to physical bandwidth K. Note that by this assumption we estimate that the available bandwidth for the proposed mechanism are at maximum, meaning that

the analysis gives the upper bound throughput of the proposed mechanism.

From Figure 6, by using ρ ($0 < \rho < 1$), the window size of the proposed mechanism just before packet losses occur is represented as $\rho K\tau$. Since the sum of the window size of both connections is $K\tau + B$ when the buffer becomes full, the window size of TCP Reno connection at that time can be described as $(1-\rho)K\tau + B$. Then, the window size of the proposed mechanism immediately after packet losses occur becomes decreased to $\rho K\tau/2$, and that of TCP Reno becomes $((1-\rho)K\tau + B)/2$. Since TCP Reno increases its window size by one packet every RTT, T, which is the time duration of one cycle, can be calculated as follows:

$$T = \frac{(1-\rho)K\tau + B}{2}\bar{\tau} \tag{23}$$

where $\bar{\tau}$ is the average value of the RTTs of the TCP connection. On the other hand, the window size of the proposed mechanism can be obtained from Equation (10) by substituting K for A as follows:

$$w(t) = \frac{w(0)e^{\epsilon t}K\tau}{w(0)(e^{\epsilon t}-1) + K\tau} \tag{24}$$

From Equations (12) and (24), we can calculate T, which equals the time it takes for the window size to increase from $\rho K\tau/2$ to $\rho K\tau$, as follows:

$$T = \frac{1}{\epsilon} ln\left(\frac{\rho}{1-\rho}\frac{K\tau - \rho K\tau/2}{\rho K\tau/2}\right) = \frac{1}{\epsilon} ln\left(\frac{2-\rho}{1-\rho}\right) \tag{25}$$

From Equations (23) and (25), we obtain the following equation:

$$\frac{(1-\rho)K\tau + B}{2}\bar{\tau} = \frac{1}{\epsilon} ln\left(\frac{2-\rho}{1-\rho}\right) \tag{26}$$

Note that the ratio of the throughput of the TCP Reno connection to that of the proposed mechanism is equal to the ratio of the areas enclosed in the x axis and each line indicating changes in the window size, as depicted in Figure 6. The area for TCP Reno, S_{reno}, is given by:

$$S_{\mathrm{reno}} = \frac{3}{4}\left\{(1-\rho)\,K\tau + B\right\}^2 \bar{\tau}$$

On the other hand, the area for the proposed mechanism, which is denoted as S_{proposed}, is calculated as follows:

$$S_{\mathrm{proposed}} = \int_0^T w(t)dt = \frac{K\tau}{\epsilon} ln\frac{2-\rho}{2(1-\rho)}$$

Finally, the average ratio of the throughput of TCP Reno to the proposal mechanism is given by:

$$\lambda = \frac{S_{\mathrm{reno}}}{S_{\mathrm{proposed}}} = \frac{\frac{3}{4}\left\{(1-\rho)\,K\tau + B\right\}^2}{\frac{K\tau}{\epsilon} ln \frac{2-\rho}{2(1-\rho)}} \tag{27}$$

From Equations (26) and (27), we can understand the relationship between the variables (ϵ, K and B) and the ratio of throughput λ. We show some numerical examples of the throughput ratio. Here we ignore the queueing delay and assume $\bar{\tau} = \tau$. Figure 7 shows changes in the throughput ratio against ϵ, where we set K = 10 [Mbps] and τ = 50 [msec]. The five lines represent the results when the buffer size B is 1/4, 1/2, 1, 2, and 4 times the bandwidth-delay product (BDP) of the bottleneck link, respectively. In Figure 8, we show the results when we set τ = 50 [msec] and B to 41 [packets] (equal to BDP when K = 10 [Mbps]), where the five lines describe the results when K = 10, 50, 100, 500, and 1000 [Mbps]. These results show that ϵ, which realizes fairness between TCP Reno and the proposed mechanism, drastically changes when we modify K and/or B. Furthermore, in some situations, especially when the buffer size is large compared with bandwidth-delay product, fairness cannot be realized by configuring ϵ. One reason is that the proposed mechanism converges its window size to $K\tau$, whereas TCP Reno continues increasing its window size until the buffer has been fully used. The congestion control algorithm of the proposed mechanism is essentially more conservative than TCP Reno. In contrast, TCP Reno has an aggressive window size control algorithm. Therefore, we cannot avoid situations where the throughput of the proposed mechanism is less than TCP Reno when they co-exist in the network. A similar discussion can also be found in the literature regarding TCP Vegas [29], and we believe it is the main reason that TCP Vegas was not successfully deployed in the Internet. In the case of TCP Reno and its variants using AIMD/MIMD policies, window size just after packet losses occur is dependent on the bottleneck link buffer size. That is, the throughput of those connections is improved as buffer size increases. However, as buffer size becomes larger, the packets within the buffer also become larger, which means that the queueing delay is also increased.

Furthermore, due to the difference between the technology evolution in memory and in link bandwidth, we are unable to prepare enough buffer for TCP connections to retain the utilization of high-speed network links in the near future [30]. That is, AIMD/MIMD-based congestion control mechanisms will fail to provide enough performance for transport layer protocols in the future Internet, and more conservative mechanisms will be needed in such situations. We believe that the proposed mechanism in this thesis is the most feasible solution. FAST TCP, originating

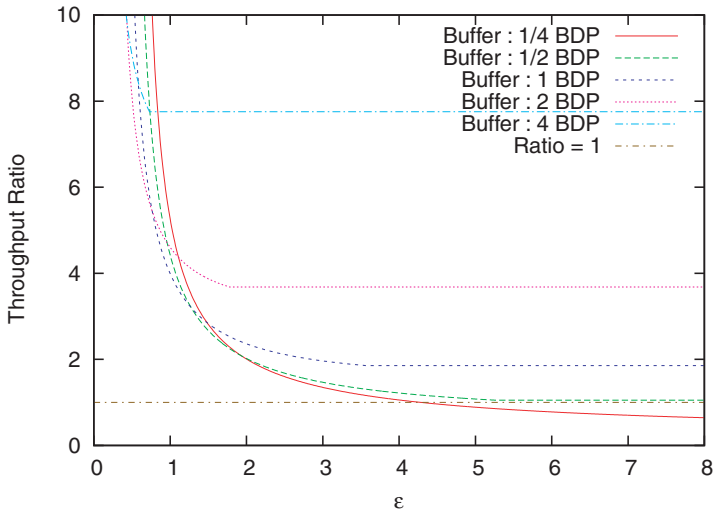


Figure 7: Ratio of throughput for various buffer sizes

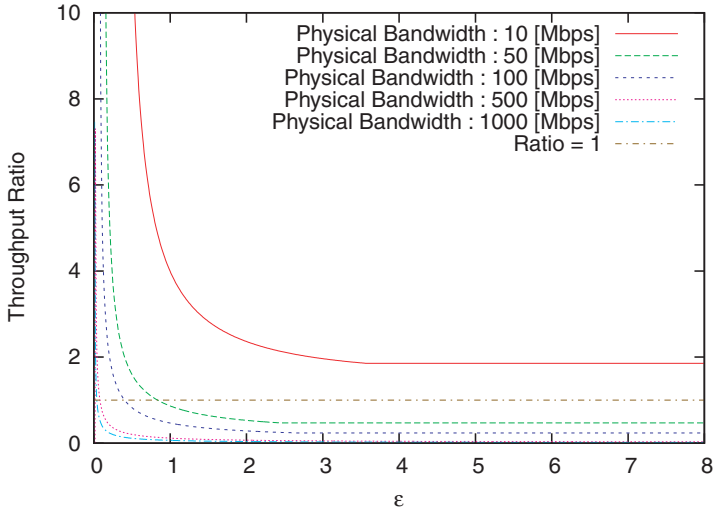


Figure 8: Ratio of throughput for various physical bandwidths

from TCP Vegas, is also a possible answer since it has a more conservative congestion control mechanism than TCP Reno. However, as described in Subsection 4.1, FAST TCP cannot provide good scalability to the number of connections sharing the bottleneck link.

5 Simulation Results

In this section, we present simulation results to evaluate the performance of the congestion control mechanism proposed in Section 3.

5.1 Simulation Settings

We use ns-2 [31] for the simulation experiments. A traditional TCP Reno, HighSpeed TCP (HSTCP) [11], Scalable TCP (STCP) [15], and FAST TCP [16] are chosen for performance comparison. In the proposed mechanism, available bandwidth information is obtained by the current ImTCP if only one connection exists. However, we directly give available bandwidth information when two or more connections exist. We also directly give the physical bandwidth information to the TCP sender. We set ϵ = 1.95 and γ = 0.9 for the proposed mechanism. The parameters in HSTCP and STCP are set to the value described in [11] and [15], respectively, and SACK option [32] is set enabled for both protocols. FAST TCP has parameter α, which should be changed according to the link bandwidth. According to the guidelines in [33] we set α = 10, 20, 50, 100, 200, 500, and 1000 for link bandwidths K = 10, 20, 50, 100, 200, 500, and 1000 [Mbps], respectively.

The network model used in the simulation is depicted in Figure 9. It consists of sender/receiver hosts, two routers, and links between the hosts and routers. $N_{\rm tcp}$ TCP connections are established between TCP sender i and TCP receiver i. For creating background traffic, we injected UDP packets at a rate of $r_{\rm udp}$ into the network, where packet size distribution follows the traffic observation results in the Internet [34]. That is, $N_{\rm tcp}$ TCP connections and an UDP flow share a bottleneck link between the two routers. The bandwidth of the bottleneck link is denoted as BW, and the propagation delay is τ. The bandwidth and the propagation delay of the access link for TCP sender i are bw_i and τ_i, respectively. We deployed the TailDrop scheme at the router buffer, and the buffer size is set equivalent to the bandwidth-delay product between sender and receiver hosts.

5.2 Fundamental Behavior

First, we confirm the fundamental behavior of the proposed mechanism with one TCP connection. Figure 10 shows the changes in the window size of TCP Reno, HSTCP, STCP, FAST TCP, and the proposed mechanism, where we set $N_{\rm tcp}$ = 1, BW = 100 [Mbps], τ = 25 [msec],

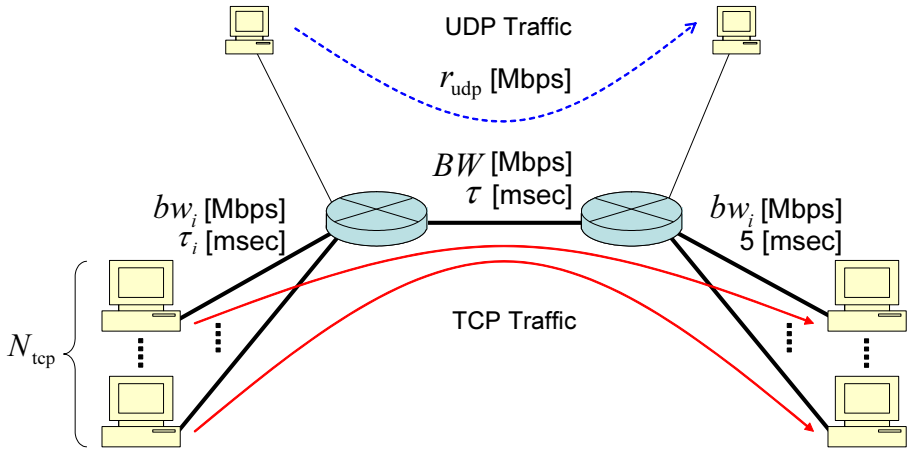


Figure 9: Network topology in simulation experiments

bw_1 = 200 [Mbps], and τ_1 = 5 [msec]. In this case, we don't inject UDP traffic into the network. This result shows that TCP Reno, HSTCP, and STCP connections experience periodic packet losses due to buffer overflow, since they continue increasing the window size until packet loss occurs. On the other hand, since the window size of FAST TCP and the proposed mechanism quickly converges to an ideal value, no packet loss occurs. Their increasing speed is much larger than HSTCP and STCP, meaning that they can more effectively utilize the link bandwidth. Furthermore, we show Figure 11, which describes the results in the case of 10 times the bandwidth of the bottleneck and access links. From these results, we observe that TCP Reno and HSTCP increase their window size slowly. But, the increasing speed of the window size of the other mechanisms remains fast regardless of the link bandwidth. Note also that HSTCP and STCP rapidly increase their window size, they cause more packet losses than in TCP Reno. In the case of Figure 10, the SACK mechanism works well, and the sender host avoids timeouts. However, in the case of Figure 11, many retransmission timeouts happen since the SACK mechanism cannot recover all of the lost packets.

5.3 Scalability to Network Bandwidth and Delay

We next investigate the scalability to the link bandwidth of the proposed mechanism by checking convergence time, defined as the time it takes for the TCP connection to utilize 99% of the link bandwidth. We set $N_{\rm tcp}$ = 1, τ_1 = 5 [msec], τ = 25 [msec], and $\tau_{\rm u}$ = 5 [msec]. Figure 12 shows

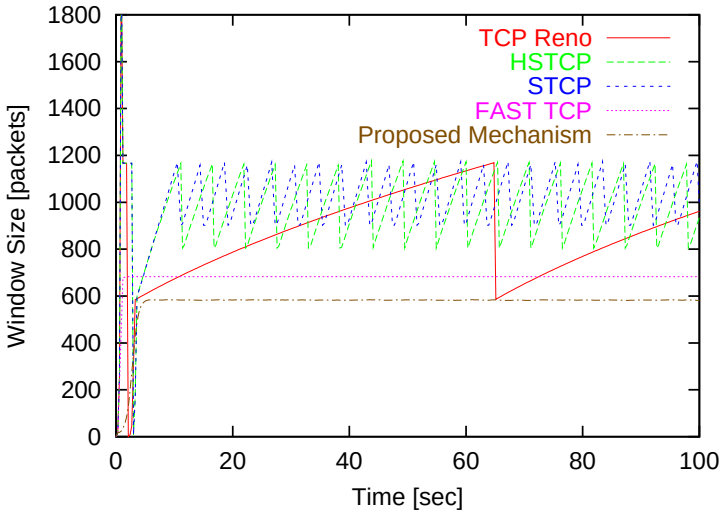


Figure 10: Changes in window size (BW=100 [Mbps])

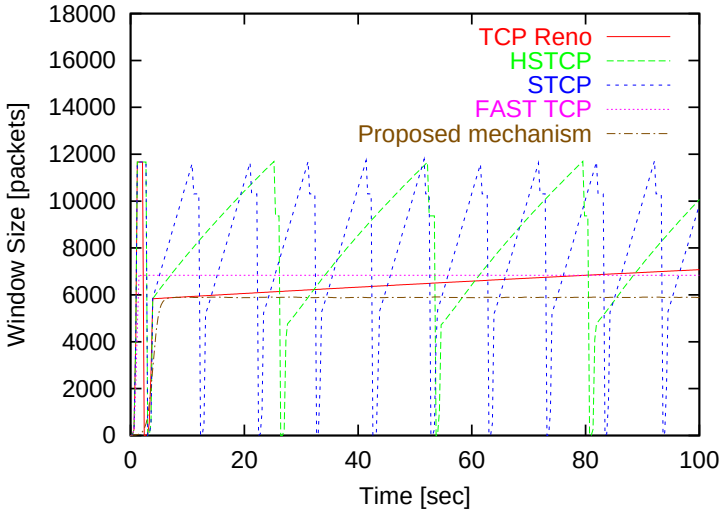


Figure 11: Change in window size (BW=1 [Gbps])

changes in the convergence time when we change BW from 10 [Mbps] to 1 [Gbps], where $r_{\rm udp}$ is set to (0.2 BW) [Mbps] and bw_1 is set equal to BW. In the figure, the average values and the 95% confidence intervals for 10 simulations experiments are shown. From this figure, we can see that the TCP Reno connection requires a large amount of time to fully utilize the link bandwidth since the increasing speed of the window size is fixed at a small value regardless of the link bandwidth. HSTCP dramatically reduces convergence time, but the larger the link bandwidth becomes, the more convergence time is required to fill the bottleneck link bandwidth. This means that HSTCP is fundamentally unable to resolve the scalability problem of TCP Reno. In the case of STCP and FAST TCP, the convergence time remains constant regardless of the link bandwidth, which is also confirmed in [15] and [16]. The proposed mechanism retains almost a constant convergence time regardless of the link bandwidth, which shows good scalability to network bandwidth.

Moreover, we investigate the scalability to the propagation delay of the proposed mechanism. We set $N_{\rm tcp}$ = 1, BW = 100 [Mbps], bw_1 = 200 [Mbps], τ_1 = 5 [msec], and $r_{\rm udp}$ = 20 [Mbps]. Figure 13 shows the changes in the convergence time when we change τ from 10 [msec] to 500 [msec]. This figure shows that the TCP Reno connection requires quite a large amount of time to fully utilize the link bandwidth since it only increases its window size by one packet per RTT. The convergence time of HSTCP and FAST TCP is less than TCP Reno. However, the more increases in propagation delay, the larger convergence time becomes. Although STCP has good scalability to link bandwidth as described in Figure 12, the convergence time increases when delay becomes larger because HSTCP, STCP, and FAST TCP increase their window size when receiving ACK packets, which depends on RTT. The proposed mechanism keeps the best scalability to the network delay. This is because, as shown in Subsection 4.1, convergence time becomes logarithmically larger against increases in delay and bandwidth.

5.4 Adaptability and Fairness

We also investigate the adaptability and fairness of the proposed mechanism by checking the effect of changes in the number of TCP connections. We set $N_{\rm tcp}$ = 5, BW = 100 [Mbps], τ = 25 [msec], bw_i = 100 [Mbps] ($1 \leq i \leq 5$), and τ_i = 5 [msec]. We don't inject UDP traffic into the network. TCP connections 1–5 join the network at 0, 100, 300, 500, and 700 [sec] and stop sending data packets at 900, 950, 1000, 1050, and 1100 [sec], respectively. Figure 14 shows changes in window size for the five TCP connections against the time in TCP Reno, HSTCP, STCP, FAST TCP, and

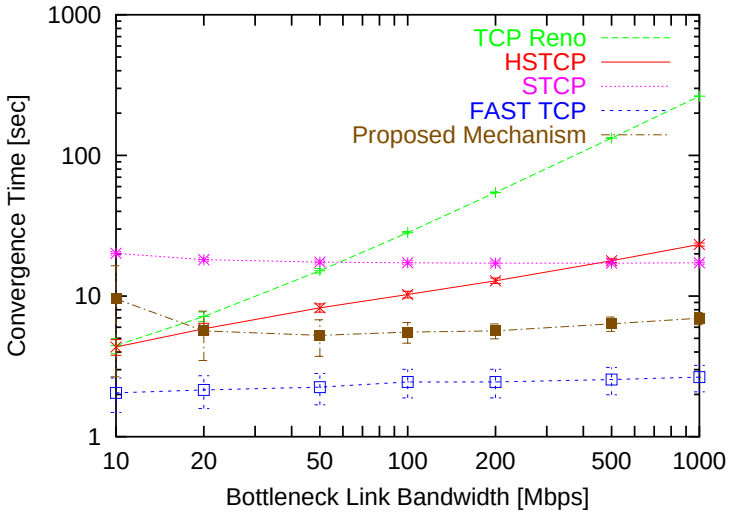


Figure 12: Changes in convergence time against bottleneck link bandwidths

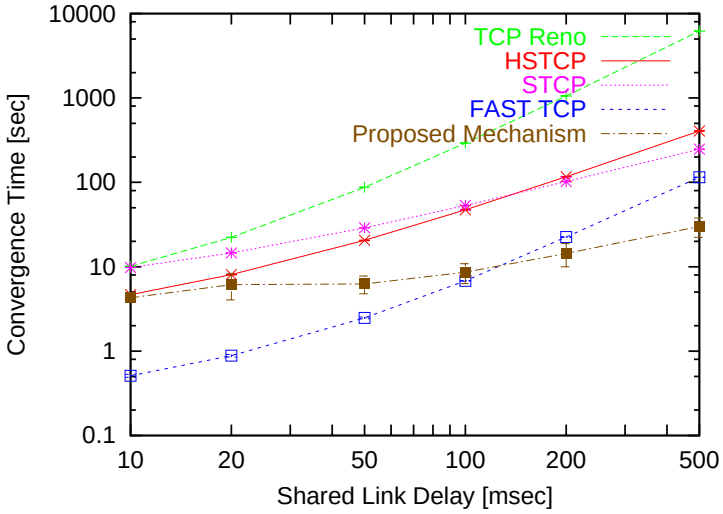
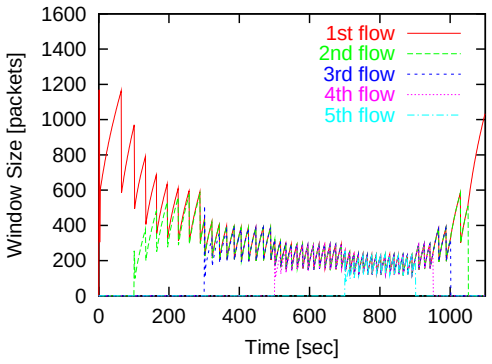


Figure 13: Changes in convergence time against bottleneck link delays

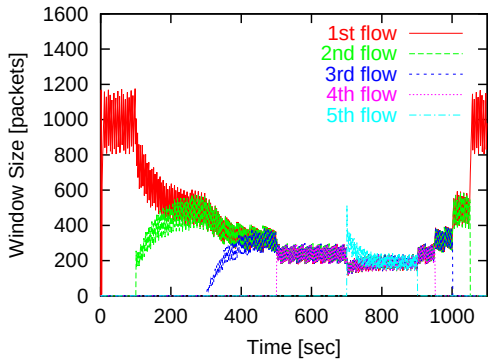
the proposed mechanism.

Figures 14(a) and 14(b) show that TCP Reno and HSTCP control their window size with the AIMD policy and realize fairness among connections by inducing periodic packet losses. From Figure 14(c), we can see that STCP cannot realize fairness among connections because it has a window size control algorithm based on MIMD policy. In Figure 14(d), we can capture the nature of FAST TCP as follows. Since it utilizes queueing delay as a congestion signal, it can adjust its window size without inducing any packet losses when a new TCP connection joins the network. However, it cannot achieve fairness among existing connections and a new connection. Although FAST TCP needs RTT information to control the window size, the new connection cannot successfully measure the minimum RTT due to the queueing delay caused by the existing connection. When a connection stops a transmission and exits from the network, the remaining connections enjoy equality throughput because the buffer becomes temporarily empty, and the existing connections can measure the precise values for minimum RTT. On the other hand, Figure 14(e) shows that the proposed mechanism converges the window sizes very quickly, and so no packet loss occurs when a new connection joins the network. Furthermore, when the TCP connection leaves the network, the proposed mechanism connections quickly fill the unused bandwidth.

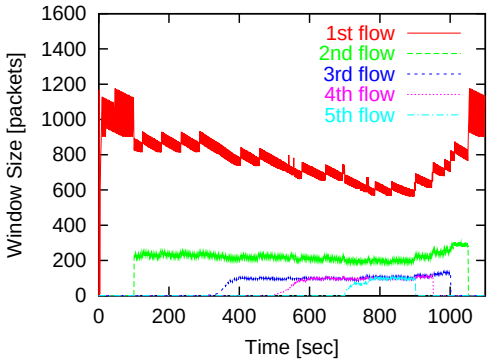
Adaptability to changes in the available bandwidth is also an important characteristic of the transport layer protocol. To confirm, we set N_{tcp} = 1, BW = 100 [Mbps], τ = 25 [msec], bw_1 = 100 [Mbps], τ_1 = 5 [msec], and change r_{udp} so that the available bandwidth of the bottleneck link is 80 [Mbps] at 0–50 [sec], 65 [Mbps] at 50–100 [sec], 50 [Mbps] at 100–150 [sec], and 80 [Mbps] at 150–200 [sec]. Figures 15 and 16 present the changes in the throughput of a TCP connection and the queue size of the bottleneck link buffer in the cases of TCP Reno, HSTCP, STCP, and the proposed mechanism. The results obviously show the effectiveness of the proposed mechanism, which gives good adaptability to the changes in the available bandwidth. Furthermore, no packet loss occurs even when the available bandwidth suddenly decreases. On the other hand, TCP Reno connections experience packet losses during simulation time, and link utilization is much lower than 100%. Although HSTCP and STCP can retain their link utilization because of the sufficient buffer, they have largely fluctuated RTTs caused by queueing delays. FAST TCP and the proposed mechanism experience no packet losses and retain their link utilization with small RTTs, but the proposed mechanism gives the smaller queue size than FAST TCP. This is one of the advantages of the proposed mechanism that uses an inline measurement technique.



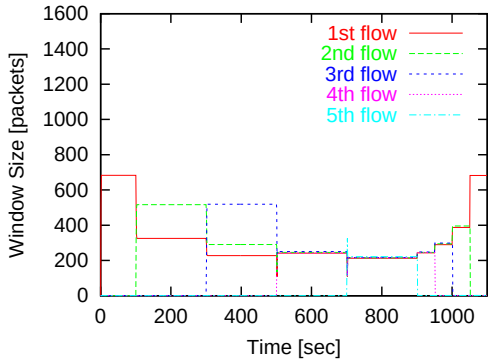
(a) TCP Reno



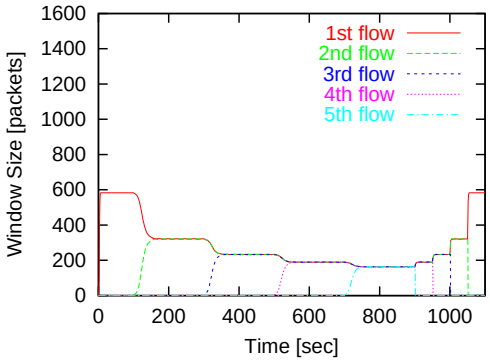
(b) HSTCP



(c) STCP



(d) FAST TCP



(e) Proposed Mechanism

Figure 14: Effect of changes in number of connections

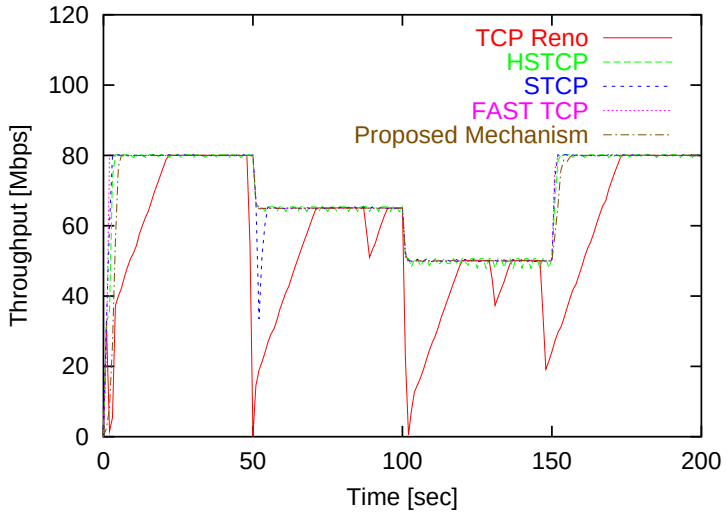


Figure 15: Adaptability to change in available bandwidth (throughput)

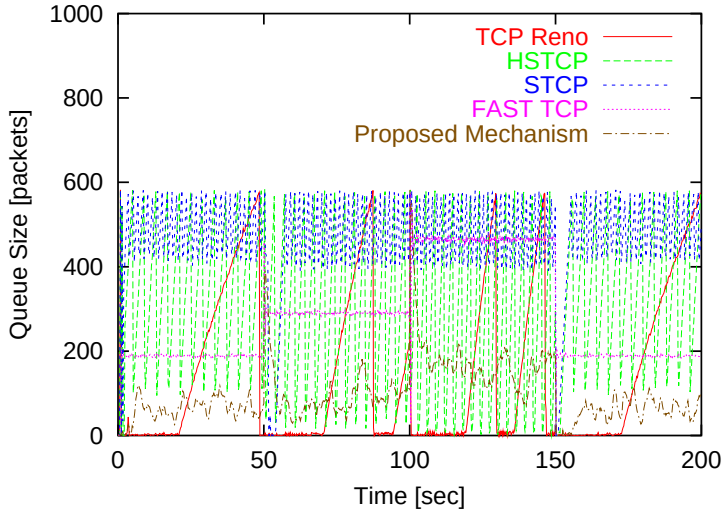


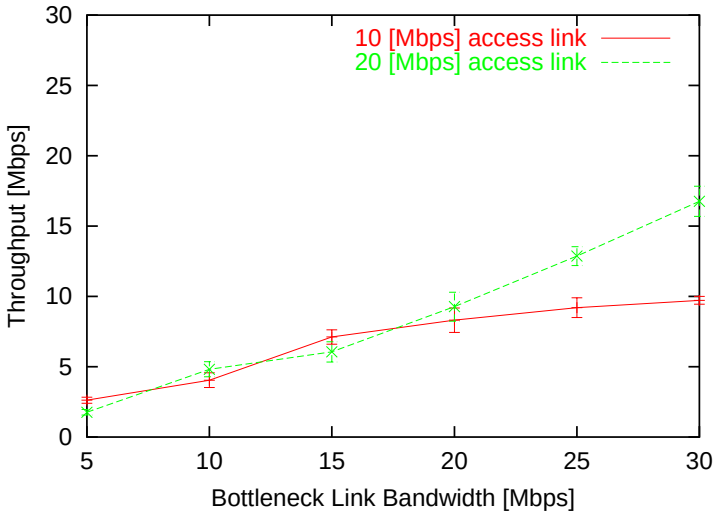
Figure 16: Adaptability to change in available bandwidth (queue size)

5.5 Effect of Heterogeneity in Physical Bandwidth

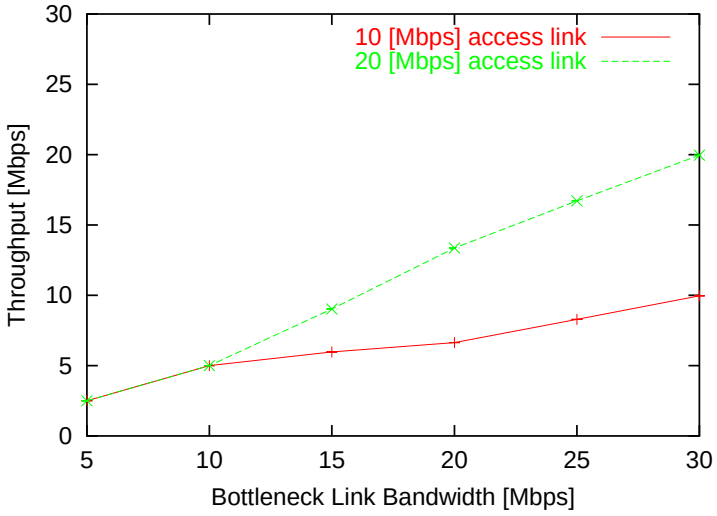
Finally, we investigate the effects of the heterogeneity of access networks such as the difference of access link bandwidth. We set $N_{\rm tcp}$ = 2, τ = 40 [msec], bw_1 = 10 [Mbps], bw_2 = 20 [Mbps], $\tau_1 = \tau_2$ = 5 [msec], and we change BW from 5 [Mbps] to 30 [Mbps]. We don't inject UDP traffic into the network. Figure 17 shows the changes in the throughput of the two TCP connections in TCP Reno and the proposed mechanism against BW. We observe from the figure that TCP Reno fairly shares the bottleneck link bandwidth regardless of the value of BW. On the other hand, the proposed mechanism shows an interesting characteristic. When $BW < bw_1$, the two TCP connections fairly share bottleneck link bandwidth. When $bw_1 < BW < bw_2$, however, the bottleneck link bandwidth is distributed proportionally to the ratio of bw_1 and bw_2. This property can be explained from the equation utilized by the proposed mechanism. By using Equation (8), the converged transmission rate for connection i, denoted by $\hat{N}_i$, which has different physical link bandwidth (K_i), can be calculated as follows:

$$\hat{N}_i = \frac{K_i}{\sum_{i=1}^{n} K_i} \cdot BW \tag{28}$$

It is satisfied under conditions of $\gamma < 1$. That is, bottleneck link bandwidth is shared proportionally to the physical bandwidth of each TCP connection. Since a physical bandwidth of the network path is defined as a bandwidth of the tightest link between TCP hosts (a sender and a receiver), the simulation results in Figure 17 match Equation (28). We argue that this characteristic is ideal for a congestion control strategy on the Internet; in the history of the Internet, the ratio of the bandwidth of access networks to backbone networks has been changing over time [35]. Compared with access networks, the amount of the resources of backbone networks are sometimes scarce and sometimes plentiful. We believe that when backbone resources are small, they should be shared fairly between users regardless of their access link bandwidth. When they are sufficient, on the other hand, they should be shared according to the access link bandwidth. The characteristics of the proposed mechanism found in Figure 17 and Equation (28) realize such a resource sharing strategy.



(a) TCP Reno



(b) Proposed Mechanism

Figure 17: Effect of different access link bandwidths

6 Conclusion

In this thesis, we proposed a new congestion control mechanism of TCP based on the inline network measurement. The proposed mechanism obtains the information of physical and available bandwidths from the inline network measurement, ImTCP. By using bandwidth information, the proposed mechanism adjusts its window size with an algorithm based on mathematical models in biophysics. Consequently, the proposed mechanism can converge its window size to an ideal value and avoid periodic packet losses experienced by TCP Reno.

Through mathematical analysis, we confirmed that the proposed mechanism has good scalability to not only link bandwidth but also propagation delay between the sender and receiver hosts; other transport layer protocols such as TCP Reno, HighSpeed TCP, Scalable TCP, and FAST TCP cannot provide such scalability. Furthermore, from the mathematical analysis results about competition between TCP Reno and the proposed mechanism, we observed, even though the realization of fairness between them is difficult, the proposed mechanism is the only solution for transport layer protocols for future high-speed networks. Furthermore, through extensive simulations, we determined that the proposed mechanism exhibits the characteristics confirmed by the analysis. Therefore, we believe that our proposed mechanism is effective regardless of network bandwidth and delay and can solve the many problems in TCP Reno and its variants.

For future work, we will confirm additional characteristics of the proposed mechanism, which include fairness among connections with different RTTs and the effect of measurement errors on the physical and available bandwidths. Implementation experiments are also important research tasks.

Acknowledgement

I am very grateful for the advice and support of my advisor, Professor Masayuki Murata of Osaka University, for his continuous feedback and for keeping me focused in my research.

All works of this thesis would not been possible without the support of Associate Professor Go Hasegawa of Osaka University. He has always given me appropriate guidance and invaluable direct advice.

I am also indebted to Associate Professor Naoki Wakamiya and Hiroyuki Ohsaki and Research Assistant Shin'ichi Arakawa, Ichinoshin Maki, and Masahiro Sasabe of Osaka University, who gave me helpful comments and feedback.

Finally, I want to say thanks to my many friends and colleagues in the Department of Information Networking of the Graduate School of Information Science and Technology of Osaka University for their support. Our conversations and work together have greatly influenced this thesis.

References

[1] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Reading, Massachusetts: Addison-Wesley, 1994.

[2] J. B. Postel, "Transmission control protocol," *Request for Comments 793*, Sept. 1981.

[3] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," *Request for Comments 1323*, May 1992.

[4] M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's loss recovery using limited transmit," *Request for Comments 3042*, Jan. 2001.

[5] E. Blanton, M. Allman, K. Fall, and L. Wang, "A conservative selective acknowledgment (SACK)-based loss recovery algorithm for TCP," *Request for Comments 3517*, Apr. 2003.

[6] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Journal of Computer Networks and ISDN Systems*, pp. 1–14, June 1989.

[7] S. Shenker, L. Zhang, and D. D. Clark, "Some observations on the dynamics of a congestion control algorithm," *ACM Computer Communication Review*, vol. 20, pp. 30–39, Oct. 1990.

[8] J. C. Hoe, "Improving the start-up behavior of a congestion control scheme of TCP," *ACM SIGCOMM Computer Communication Review*, vol. 26, pp. 270–280, Oct. 1996.

[9] L. Guo and I. Matta, "The War Between Mice and Elephants," *Technical Report BU-CS-2001-005*, May 2001.

[10] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla, "The impact of multihop wireless channel on TCP throughput and loss," in *Proceedings of IEEE INFOCOM 2003*, Apr. 2003.

[11] S. Floyd, "HighSpeed TCP for large congestion windows," *Request for Comments 3649*, Dec. 2003.

[12] C. Casetti, M. Gerla, S. Mascolo, M.Y.Sanadidi, and R. Wang, "TCP Westwood: End-to-end congestion control for wired/wireless networks," *Wireless Networks Journal*, vol. 8, pp. 467–479, 2002.

[13] K. Xu, Y. Tian, and N. Ansari, "TCP-Jersey for wireless IP communications," *IEEE Journal on Selected Areas in Communications*, vol. 22, pp. 747–756, May 2004.

[14] E. H.-K. Wu and M.-Z. Chen, "JTCP: Jitter-based TCP for heterogeneous wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 22, pp. 757–766, May 2004.

[15] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," in *proceedings of PFLDnet '03: workshop for the purposes of discussion*, Feb. 2003.

[16] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: motivation, architecture, algorithms, performance," in *Proceedings of IEEE INFOCOM 2004*, Mar. 2004.

[17] B. Melander, M. Bjorkman, and P. Gunningberg, "A new end-to-end probing and analysis method for estimating bandwidth bottlenecks," in *Proceedings of IEEE GLOBECOM 2000*, Nov. 2000.

[18] M. Jain and C. Dovrolis, "End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput," in *Proceedings of ACM SIGCOMM 2002*, Aug. 2002.

[19] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell, "pathChirp: Efficient available bandwidth estimation for network paths," in *Proceedings of NLANR PAM2003*, Apr. 2003.

[20] R. L. Carter and M. E. Crovella, "Measuring bottleneck link speed in packet-switched networks," Tech. Rep. BU-CS-96-006, Boston University Computer Science Department, Mar. 1996.

[21] C. Dovrolis, P. Ramanathan, and D. Moore, "What do packet dispersion techniques measure?," in *Proceedings of IEEE INFOCOM 2001*, Apr. 2001.

[22] V. Jacobson, "Pathchar - a tool to infer characteristics of internet paths." available from `http://www.caida.org/tools/utilities/others/pathchar/`, Apr. 1997.

[23] M. L. T. Cao, "A study on inline network measurement mechanism for service overlay networks," Master's thesis, Graduate School of Information Science and Technology, Osaka University, Feb. 2004.

[24] M. L. T. Cao, G. Hasegawa, and M. Murata, "Available bandwidth measurement via TCP connection," in *Proceedings of IFIP/IEEE MMNS 2004*, Oct. 2004.

[25] M. L. T. Cao, G. Hasegawa, and M. Murata, "A merged inline measurement method for capacity and available bandwidth," to be presented at *NLANR PAM 2005*, Mar. 2005.

[26] L. S. Brakmo, S. W.O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proceedings of ACM SIGCOMM'94*, pp. 24–35, Oct. 1994.

[27] J. D. Murray, *Mathematical Biology I: An Introduction*. Springer Verlag Published, 2002.

[28] E. Blanton and M. Allman, "On making TCP more robust to packet reordering," *ACM Computer Communication Review*, vol. 32, pp. 20–30, Jan. 2002.

[29] K. Kurata, G. Hasegawa, and M. Murata, "Fairness comparisons between TCP Reno and TCP Vegas for future deployment of TCP Vegas," in *Proceedings of IEEE INET 2000*, July 2000.

[30] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," in *Proceedings of ACM SIGCOMM 2004*, Aug. 2004.

[31] The VINT Project, "UCB/LBNL/VINT network simulator - ns (version 2)." available from `http://www.isi.edu/nsnam/ns/`.

[32] M. Mathis, "TCP selective acknowledgement options," *Request for Comments 2018*, Oct. 1996.

[33] C. Jin, D. X. Wei, and S. H. Low, "Internet draft: FAST TCP for high-speed long-distance networks," *Internet draft draft-jwl-tcp-fast-01.txt*, June 2003.

[34] A. Technologies, "Mixed packet size throughput." available from `http://advanced.comms.agilent.com/n2x/docs/journal/JTC_003.html`.

[35] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield, "QoS's downfall: At the bottom, or not at all!," in *Proceedings of ACM SIGCOMM 2003 Workshop on Revisiting IP QoS (RIPQOS)*, Aug. 2003.