Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Communications xx (xxxx) 1–13
computer communications
www.elsevier.com/locate/comcom

ImTCP: TCP with an inline measurement mechanism for available bandwidth

Cao Le Thanh Man*, Go Hasegawa, Masayuki Murata

Graduate School of Information Science and Technology, Osaka University, Japan

Received 14 July 2005; accepted 14 July 2005

Abstract

We introduce a novel mechanism for actively measuring available bandwidth along a network path. Instead of adding probe traffic to the network, the new mechanism exploits data packets transmitted in a TCP connection (inline measurement). We first introduce a new bandwidth measurement algorithm that can perform measurement estimates quickly and continuously and is suitable for inline measurement because of the smaller number of probe packets required and the negligible effect on other network traffic. We then show how the algorithm is applied in RenoTCP through a modification to the TCP sender only. We call the modified version of RenoTCP that incorporates the proposed mechanism *ImTCP* (Inline measurement TCP). The ImTCP sender adjusts the transmission intervals of data packets, then estimates available bandwidth of the network path between sender and receiver utilizing the arrival intervals of ACK packets. Simulations show that the new measurement mechanism does not degrade TCP data transmission performance, has no effect on surrounding traffic and yields acceptable measurement results in intervals as short as some RTTs (round-trip times). We also give examples in which measurement results help improving TCP performance.

Keywords: Available bandwidth; Inline measurement; TCP

1. Introduction

Information concerning bandwidth availability in a network path plays an important role in adaptive control of the network. Many research on measuring available bandwidth have been done so far. Available bandwidth can be measured at routers within a network [1]. This approach may require a considerable change to network hardware and is suitable for network administrators only. Some passive measurement tools can collect traffic information at some end hosts for performance measurements [2], but this approach requires a relatively long time for data collection and bandwidth estimation. Exchanging probe traffic between two end hosts to find the available bandwidth along a path (an active measurement) seems the more realistic approach and has attracted much recent research [3–7].

Sending extra traffic into the network is the common weakness in all active available bandwidth measurement tools. Depending on the algorithm used, the amount of required probe traffic differs. According to one study [7], Pathload [4] generated between 2.5 and 10 MB of probe traffic per measurement. Newer tools have succeeded in reducing this. The average per-measurement probe traffic generated by IGI [6] is 130 KB and by Spruce [7] is 300 KB. A few KB of probe traffic for a single measurement is a negligible load on the network. But for routing in overlay networks, or adaptive control in transmission protocols, these measurements may be repeated continuously and simultaneously from numerous end hosts. In such cases, the few KB of per-measurement probes will create a large amount of traffic that may damage other data transmission in the network as well as degrade the measurement itself.

We propose an active measurement method that does not add probe traffic to the network, with the idea of 'plugging' the new measurement mechanism into an active TCP connection (inline measurement). That is, data packets and ACK packets of a TCP connection are utilized for

* Corresponding author.
E-mail addresses: mlt-cao@ist.osaka-u.ac.jp (C.L.T. Man), hasegawa@ist.osaka-u.ac.jp (G. Hasegawa), murata@ist.osaka-u.ac.jp (M. Murata).

0140-3664/$ - see front matter © 2005 Published by Elsevier B.V.
doi:10.1016/j.comcom.2005.07.014

the measurement, instead of probe packets. This method has the advantage of requiring no extra traffic to be sent to the network.

We first introduce a measurement algorithm suitable for inline network measurement that generates periodic measurement results at short intervals, on the order of several RTTs. The key idea in measuring rapidly is to limit the bandwidth measurement range using statistical information from previous measurement results. This is done rather than searching from 0 bps to the upper limit of the physical bandwidth with every measurement as existing algorithms do [4,5]. By limiting the measurement range, we can avoid sending probe packets at an extremely high rate and keep the number of probe packets small.

We then introduce ImTCP (Inline measurement TCP), a Reno-based TCP that includes the proposed algorithm for inline network measurement described above. When a sender transmits data packets, ImTCP first stores a group up to several packets in a queue and subsequently forwards them at a transmission rate determined by the measurement algorithm. Each group of packets corresponds to a probe stream. Then, considering ACK packets as echoed packets, the ImTCP sender estimates available bandwidth according to the algorithm. To minimize transmission delay caused by the packet store-and-forward process, we introduce an algorithm using the RTO (round trip timeout) calculation in TCP to regulate packet storage time in the queue. We evaluate the inline measurement system using simulation experiments. The results show that the proposed algorithm works with the window-based congestion control algorithm of TCP without degrading transmission throughput.

Measurement results of ImTCP can be passed to higher network layer and used for optimal route selection [8] in service overlay networks, in network topology design or in isolating fault locations [9]. Besides, ImTCP can use such bandwidth information to optimize link utilization or improve transmission performance of itself. We present two examples of the second usage. In background mode, ImTCP uses the results of bandwidth availability measurements to prevent its own traffic from degrading the throughput of other traffic. This allows a prioritization of other traffic sharing the network bandwidth. In full-speed mode, ImTCP uses measurement results to keep its transmission rate close to the measured value necessary for optimum utilization of the available network bandwidth. This mode is expected to be used in wireless and high-speed networks where traditional TCP cannot use the available bandwidth effectively.

The remainder of this paper is organized as follows. In Section 2, we discuss related works concerning inline measurement. In Section 3, we introduce our proposed algorithm for inline network measurement and ImTCP. In Section 4, we evaluate ImTCP performance. In Section 5, we introduce two examples of congestion window control mechanisms for ImTCP. Finally in Section 6, we present concluding remarks and discuss future projects.

2. Related research on inline network measurement

The idea of inline measurement has previously appeared in traditional TCP. To some extent, traditional TCP can be considered a tool for measuring available bandwidth because of its ability to adjust the congestion window size to achieve a transmission rate appropriate to the available bandwidth. One version of TCP, TCP Vegas [10], also measures the packet transmission delay. There are, in addition, other tools that convert the TCP data transmission stack into network measurement tools; Sting [11] (measuring packet loss) and Sprobe [12] (measuring capacity in a bottleneck link) are typical examples.

As for the measurement of available bandwidth in an active TCP connection, there is some related research. Bandwidth estimation in traditional TCP (Reno TCP) is insufficient and inaccurate because it is a measure of used bandwidth, not available bandwidth. Especially in networks where the packet loss probability is relatively high, TCP tends to fail at estimating available bandwidth. Moreover, the TCP sender window size often does not accurately represent the available bandwidth due to the nature of the TCP congestion control mechanism. The first TCP measurement algorithm to improve accuracy used a passive method in which the sender checks ACK arrival intervals to infer available bandwidth proposed by Hoe [13]. It is a simple approach based on the Cprobe [3] algorithm. A similar technique is used in TCP Westwood [14] where the sender also passively observes ACK packet arrival intervals to estimate bandwidth, but the results are more accurate due to a robust calculation. Another study in Ref. [15] proposes TCP-Rab, a TCP with an inline measurement method that based on the receiver. The receiver calculates the available bandwidth from the arrival rate of TCP segments and informs the sender, so that the sender can perform a measurement-based congestion window control mechanism. The approach estimates the bandwidth better than Westwood, because it can eliminate noise caused by the fluctuation of ACK packets' transmission times. However, because these methods are all passive measurements, changes in available bandwidth cannot be detected quickly. Especially, when the available bandwidth increases suddenly, the TCP data transmission rate cannot adjust as rapidly and needs time to ramp up because of the self-clocking behavior of TCP. Meanwhile, as transmission proceeds at a rate lower than the available bandwidth, the measurement algorithm yields results lower than the true value.

Our proposed algorithm uses an active approach for inline measurement. That is, the sender TCP does not only observe ACK packet arrival intervals, but also actively adjusts the transmission interval of data packets. The sender thus collects more information for a measurement and improved accuracy can be expected. Moreover, the proposed mechanism requires a modification of the TCP sender only, incurring the same deployment cost as the approaches of [13,14].

3. ImTCP: TCP with inline network measurement

3.1. Overview

We implement a program for inline network measurement in the sender program of RenoTCP to create ImTCP. The program locates at the bottom of TCP layer, as shown in Fig. 1. When a new TCP data packet is generated at the TCP layer and is ready to be transmitted, it is stored in an intermediate FIFO buffer (hereafter called the ImTCP buffer) before being passed to the IP layer. The timing at that the packets are passed to the IP layer is controlled by the program. When ImTCP performs a measurement, the program adjusts the transmission intervals of some packets according to the measurement algorithm. When ImTCP is not performing a measurement, it passes all TCP data packets arriving at the buffer immediately to the IP layer. On the other hand, when an ACK packet arrives at the sender host, the measurement program records the arrival time for measurement then passes the ACK packets to the TCP layer for TCP protocol processing.

3.2. Proposed measurement algorithm

The program adjusts the transmission intervals of packets to form packet streams that are group packets sent at one time, for the measurements. The measurements are performed repeatedly.

In every measurement, a search range is introduced for searching the value of the available bandwidth. Search range $I=(B_l, B_u)$ is a range of bandwidth which is expected to include the current value of the available bandwidth. The proposed measurement algorithm searches for the available bandwidth only within the given search range. The minimum value of B_l, the lower bound of the search range, is 0, and the maximum value of B_u, the upper bound, is equal to the physical bandwidth of the link directly connected to the sender host. By introducing the search range, sending probe packets at an extremely high rate, which seriously affects other traffic can be avoided. The number of probe packets for the measurement can also be kept quite small. As discussed later herein, even when the value of the available bandwidth does not exist within the search range, the correct value can be found in a few measurements. The following are the steps of the proposed algorithm for one measurement of the available bandwidth A:

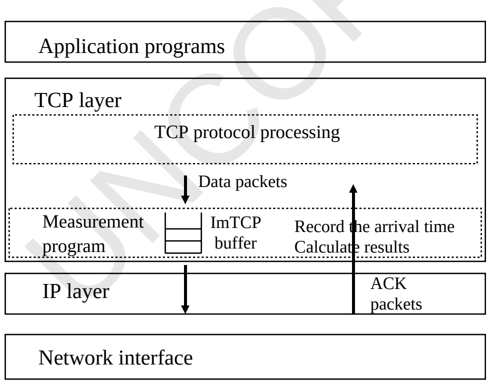


Fig. 1. Placement of measurement program at ImTCP sender.

3.2.1. Set initial search range

First, the program send a packet stream according to the Cprobe algorithm [3] to find a very rough estimation of the available bandwidth. We set the search range to $(A_{cprobe}/2, A_{cprobe})$, where A_{cprobe} is the result of the Cprobe test.

3.2.2. Divide the search range

The search range is divided into k sub-ranges $I_i=(B_{i+1}, B_i)$ $(i=1,2\ldots k)$. All sub-ranges have the identical width of the bandwidth. That is,

$$B_i = B_u - \frac{B_u - B_l}{k}(i-1) \quad (i = 1, \ldots, k+1)$$

As k increases, the results of Steps 4 and 6 become more accurate, because the width of each sub-range becomes smaller. However, a larger number of packet streams is required, which results in an increase in the number of used packets and the measurement time.

3.2.3. Send packet streams and check increasing trend

For each of k sub-ranges, a packet stream i $(i=1\ldots k)$ is sent. The transmission rates of the stream's packets vary to cover the bandwidth range of the sub-range. We denote the jth packet of the packet stream i as $P_{i,j}$ $(1\le j\le N$, where N is the number of packets in a stream) and the time at which $P_{i,j}$ is sent from the sender host as $S_{i,j}$, where $S_{i,1}=0$. Then $S_{i,j}$ $(j=2\ldots N)$ is set so that the following equation is satisfied:

$$\frac{M}{S_{i,j}-S_{i,j-1}} = B_{i+1} + \frac{B_i - B_{i+1}}{N-1}(j-1)$$

where M is the size of the probe packet. Fig. 2 shows the relationship between the search range, the sub-ranges and

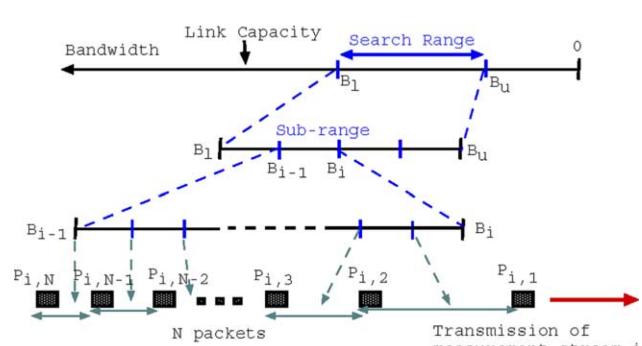


Fig. 2. Relationship of search range, sub-ranges, streams, and probe packets.

the packet streams. In the proposed algorithm, packets in a stream are transmitted with different intervals, for this reason the measurement result may not be as accurate as the Pathload algorithm [4], in which all packets in a stream are sent with identical intervals. However, the proposed algorithm can check a wide range of bandwidth with one stream, whereas the Pathload checks only one value of the bandwidth with one stream. This reduces the number of probe packets and the time required for measurement. By this mechanism, the measurement speed is improved at the expense of measurement accuracy.

The program then observe $R_{i,j}$, the time the ACK of packet $P_{i,j}$ arrives at the sender host, where $R_{i,1}=0$. We calculate the transmission delay $D_{i,j}$ of $P_{i,j}$ using the function $D_{i,j}=R_{i,j}-S_{i,j}$. We then check if an increasing trend exists in the transmission delay $(D_{i,j}-D_{i,j-1})$ $(2\le j\le N)$ according to the algorithm used in Ref. [4]. As explained in Ref. [4], the increasing trend of transmission delay in a stream indicates that the transmission rate of the stream is larger than the current available bandwidth of the network path.

Let T_i be the increasing trend of stream i as follows:

$$T_i=\begin{cases}1 & \text{increasing trend in stream } i\\ -1 & \text{no increasing trend in stream } i\\ 0 & \text{unable to determine}\end{cases}$$

As i increases, the rate of stream i decreases. Therefore, T_i is expected to be 1 when i is sufficiently small. On the other hand, when i becomes large, T_i is expected to become -1. Therefore, when neither of the successive streams m or $m+1$ have an increasing trend ($T_m=T_{m+1}=-1$), the remaining streams are expected not to have increasing trends ($T_i=-1$ for $m+2\le i\le k$). Therefore, the program stops sending the remaining streams in order to speed up the measurement.

3.2.4. Choose a sub-range

Based on the increasing trends of all streams, the algorithm chooses a sub-range, which is most likely to include the correct value of the available bandwidth. First, it finds the value of a $(0\le a\le k+1)$, which maximizes $\left(\sum_{j=0}^{a}T_j-\sum_{j=a+1}^{k}T_j\right)$. If $1\le a\le k$, it determine the sub-range I_a is the most likely candidate of the sub-range which includes the available bandwidth value. That is, as a result of the above calculation, I_a indicates the middle of streams which have increasing trends and those which do not. If $a=0$ or $a=k+1$, on the other hand, the algorithm decides that the available bandwidth does not exist in the search range $(B_l,\ B_u)$. The algorithm determines that the available bandwidth is larger than the upper bound of the search range when $a=0$, and that when $a=k+1$ the available bandwidth is smaller than the lower bound of the search range.

In this way, the algorithm finds the sub-range which is expected to include the available bandwidth according to the increasing trends of the packet streams.

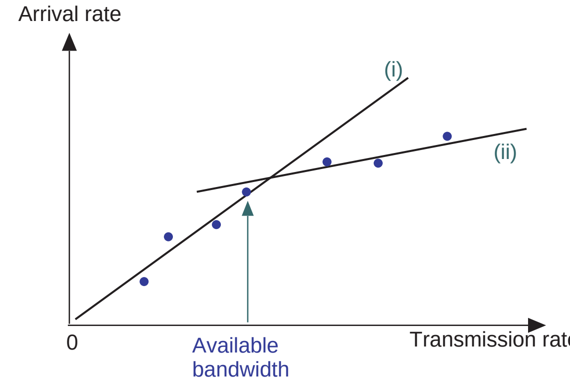


Fig. 3. Finding the available bandwidth within a sub-range.

3.2.5. Calculate the available bandwidth

The algorithm then derives the available bandwidth A from the sub-range I_a chosen by Step 4. It first determines the transmission rate and the arrival rate of the packet $P_{a,j}$ $(j=2\ldots N)$ as $\frac{M}{S_{a,j}-S_{a,j-1}}$, $\frac{M}{R_{a,j}-R_{a,j-1}}$,respectively. It then approximates the relationship between the transmission rate and the arrival rate as two straight lines using the linear regression method, as shown in Fig. 3. Since, it determines that the sub-range I_a includes the available bandwidth, the slope of line (i) which consists of small transmission rates is nearly 1 (the transmission rate and the arrival rate are almost equal), and the slope of line (ii) which consists of larger transmission rates is smaller than 1 (the arrival rate is smaller than the transmission rate). Therefore, it determines that the highest transmission rate in line (i) is the value of the available bandwidth.

On the other hand, when the algorithm has determined that the available bandwidth value does not exist in the search range (B_l, B_u) in Step 4, it temporarily set the value of available bandwidth as follows:

$$A=\begin{cases}B_l & a=0\\ B_u & a=k+1\end{cases}$$

3.2.6. Create a new search range

When the program have found the value of the available bandwidth from a sub-range I_a in Step 5, we accumulate the value as the latest statistical data of the available bandwidth. The next search range (B'_l, B'_u) is calculated as follows:

$$B'_l=A-\max\left(1.96\frac{S}{\sqrt{q}},\frac{B_m}{2}\right)$$

$$B'_u=A+\max\left(1.96\frac{S}{\sqrt{q}},\frac{B_m}{2}\right)$$

where S is the variance of stored values of the available bandwidth and q is the number of stored values. Thus, we use the 95% confidential interval of the stored data as the width of the next search range, and the current available bandwidth is used as the center of the search range. B_m is the lower bound of the width of the search range, which is used to prevent the range from being too small. When no accumulated data exists (when the measurement has just started or just after the accumulated data is discarded), we use the same search range as that of the previous measurement.

On the other hand, when we cannot find the available bandwidth within the search range, it is possible to consider that the network status has changed greatly. Therefore, we discard the accumulated data because this data becomes unreliable as statistical data. In this case, the next search range (B'_l, B'_u) is set as follows:

$$B'_l = \begin{cases} B_l & a=0 \\ B_l - \dfrac{B_u - B_l}{2} & a=k+1 \end{cases} \qquad B'_u = \begin{cases} B_u + \dfrac{B_u - B_l}{2} & a=0 \\ B_u & a=k+1 \end{cases}$$

This modification of the search range is performed in an attempt to widen the search range in the possible direction of the change of the available bandwidth.

By this statistical mechanism, we expect the measurement algorithm to behave as follows: when the available bandwidth does not change greatly over a period of time, the search range becomes smaller and more accurate measurement results can be obtained. On the other hand, when the available bandwidth varies greatly, the search range becomes large and the measurement can be restarted from the rough estimation. That is, the proposed algorithm can give a very accurate estimation of the available bandwidth when the network is stable, and a rough but rapid estimate can be obtained when the network status changes.

3.3. Packet storing mechanism

The measurement algorithm uses previous measurement results to determine a search range for the next measurement. Therefore, it is natural that only one measurement operation should be performed for one RTT. If the TCP window size is sufficiently large, we can perform multiple measurements for one RTT by introducing a quite complex mechanism. However, many difficulties must be overcome to accomplish this, including interaction of measurement tasks, delays caused by multiple streams. We therefore decided that ImTCP should perform at most one measurement operation per RTT. One RTT is long enough for ImTCP to recover the transmission rate after a measurement.

The measurement program dynamically adapts to changes in the TCP window size. It stores no data packets when the current window size is smaller than the number of packets required for a measurement stream. This is because

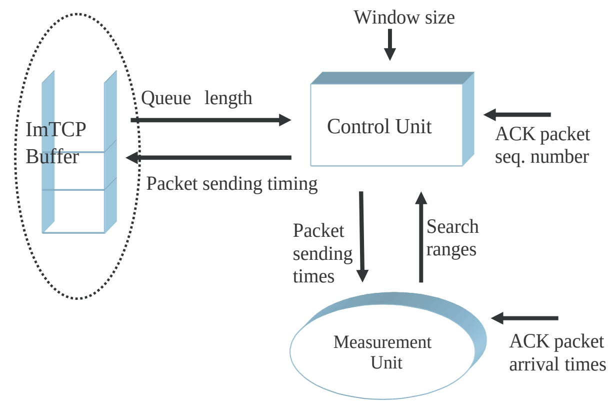


Fig. 4. Structure of the measurement program.

the TCP sender cannot transmit a number of data packets larger than the window size. On the other hand, when the window size is sufficiently large, the program creates all streams required for a measurement in each RTT.

Fig. 4 shows the structure of the measurement program. It consists of three units. The ImTCP Buffer unit stores TCP data packets and passes each packet to the IP layer under control of the Control unit. It informs the Control unit when a new TCP packet arrives. The Control unit determines when to send the packets stored in the buffer. Details of the Measurement unit were introduced in Section 3.2.

Here, we explain the operation of the Control unit. The Control unit has four functional states, STORE PACKET, PASS PACKET, SEND STREAM and EMPTY BUFFER, as shown in Fig. 5. The Control unit is initially in the STORE PACKET state. In what follows, we describe the detailed behaviors of the Control unit in each state;

1. STORE PACKET state
   - Start storing packets for the creation of measurement streams. Set the packet storing timer to end packet storing after certain length of time T. The timer value T is discussed in Section 3.4.
   - Go to the SEND STREAM state if the number of stored packets equals to m. The value of m is discussed in Section 3.4.

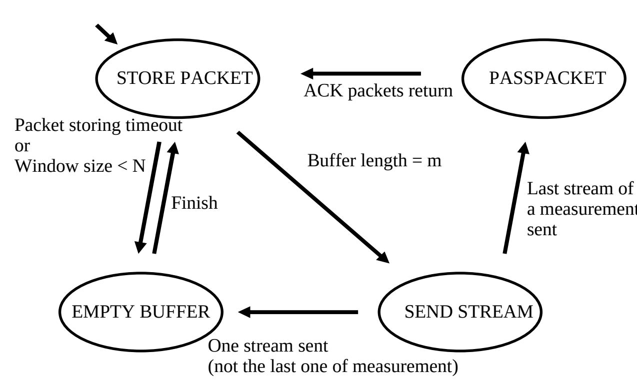


Fig. 5. State transition in the Control unit.

- Go to the EMPTY BUFFER state if the current TCP window size becomes smaller than N or the packet storing timer expires. N is the number of packets needed to create a measurement stream.

2. EMPTY BUFFER state
   - Pass currently stored packets to the IP layer until the buffer becomes empty.
   - Return to the STORE PACKET state.
3. SEND STREAM state
   - Send a measurement stream. The transmission rate of the stream is determined according to the measurement algorithm. During stream transmission, packets arriving at the buffer are stored in the ImTCP buffer.
   - After the transmission of the stream, if the stream is the last of a measurement, go to PASS PACKET state, if not, go to the EMPTY BUFFER state.
4. PASS PACKET state
   - Pass every packet in the buffer immediately to the IP layer.
   - Go to the STORE PACKET state when all ACK packets of the transmitted measurement streams have arrived at the sender.

3.4. Parameter settings

3.4.1. Number of packets required to start a measurement stream (m)

The timing for sending packets in a measurement stream is determined by the measurement algorithm. If N packets were stored prior to the beginning of transmission, the long storage time would slow the TCP transmission speed. Instead, transmission begins when only a partial number of packets (m out of N packets) have arrived in the ImTCP buffer. The timing is such that the former part of the stream is being transmitted as the latter part of the stream is still arriving at the buffer, and the latter packets are expected to arrive in time for transmission. Thus, we reduce the effect of the packet storing mechanism on TCP transmission.

If we set m to a very small value, the latter part of the stream will not be available when the former part of the stream has already been transmitted, in which case the stream transmission fails. Therefore, m must be large enough to ensure successful transmission of the measurement stream, but no larger. The algorithm for determining m is given below. In the algorithm, m is adjusted according to whether or not transmission of the previous measurement streams was successful.

- Set $m=N$ initially. The minimum of m is 2, and the maximum of m is N.
- If F successive measurements are completed successfully, and m is greater than its minimum of 2, then decrement m by 1. We set F to 2.
- If a stream creation fails, and m is less than its maximum of N, then decrement m by 1 and create the stream again.

3.4.2. Packet storing timer (T)

We avoid degrading the TCP transmission speed, caused by storing data packets before they are passed to the IP layer, by appropriately setting a timer to stop the creation of a stream. Obviously, there is a trade-off between measurement frequency and TCP transmission speed when choosing the timer value. That is, for large timer values, the program can create measurement streams frequently so measurement frequency increases. In this case, however, because TCP data packets may be stored in the intermediate buffer for a relatively long period of time, TCP transmission speed may deteriorate. Following is an example. An application temporarily stops sending data, but the measurement program is still waiting for more packets to form a measurement stream. There is no new data packet arriving at the ImTCP buffer so the packets currently in the buffer are delayed until the application sends new data. In this situation, if the application does not send data within 1 s, the TCP timeout will occur.

On the other hand, for small timer values, the program may frequently fail to create packet streams, leading a low frequency of measurement success. In the following discussion, we derive the appropriate value for the packet storing timer by applying an algorithm similar to the RTO calculation in TCP [16].

If we assume a normal distribution of packet RTTs with average A_{RTT} and variance D_{RTT}, A_{RTT} and D_{RTT} can be inferred from the TCP timeout function [16]. We use the following notation;

- X: RTT of a TCP data packet
- Y: The time since the first of N successive data packets is sent until the ACK of the last packet arrives at the sender
- Z: The time necessary for N successive ACK packets to arrive at the sender

We illustrate X, Y and Z in Fig. 6. We need to know the distribution of Z to determine the appropriate value for

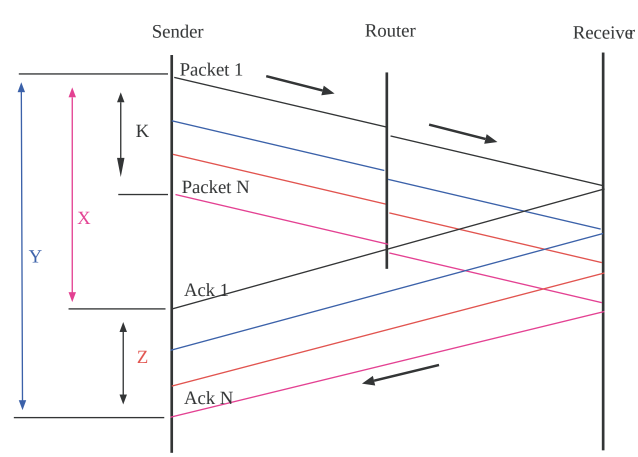


Fig. 6. Packets transmission times in TCP.

the packet storing timer. From Fig. 11, we can see that:

$$Z = Y - X \tag{1}$$

From the assumption mentioned above, X has a normal distribution $N(A_{\mathrm{RTT}}, D_{\mathrm{RTT}})$. Note that Y is the period of time from sending the first packet until the last packet is sent (we denote the length of this period as K) plus the RTT of the last packet. That is, we can conclude that the distribution of Y is $N(A_{\mathrm{RTT}}+K,\ D_{\mathrm{RTT}})$. From Eq. (1) we then obtain the distribution of Z, as $N(K, 2\cdot D_{\mathrm{RTT}})$.

Here, we provide a simple estimate of K. In a TCP flow, due to the self-clocking phenomenon, the TCP packet transmission rate is a rough estimate of the available bandwidth of the network link. The average time needed to send N successive TCP data packets is

$$K = \frac{M}{A}(N-1) \tag{2}$$

where M is the packet size and A is the value of available bandwidth which can obtain from the measurement results. From the distribution of Z and Eq. (2), we determine the waiting time for N ACK packets as below:

$$T = \frac{M}{A}(N-1) + 4D_{\mathrm{RTT}}$$

Using this value for the timer, the probability of successfully collecting N packets reaches approximately 98% due to the characteristics of the normal distribution. Thus, we are using a relatively short timer length that reduces additional processing delays caused by the measurement program but provides a high probability of collecting a sufficient number of packets for creating measurement streams.

3.5. Other issues

3.5.1. Effect of delayed ACK option

When a TCP receiver uses the delayed ACK option, it sends only one ACK packet for every two data packets. In this case, the proposed algorithm does not work properly, since it assumes the receiver host will send back a probe packet for each received packet. To solve this problem, Step 3 in Section 3.2 of the proposed algorithm should be changed so that intervals of three packets are used rather than intervals of two packets. That is, we calculate the transmission delay $(D_{i,2j'+2} - D_{i,2j'})$ $(1 \le j' \le \lfloor N/2 \rfloor)$ for the probing packets in stream i in order to check its increasing trend. This modification has almost the same effect as halving the number of packets in one stream, resulting in a degradation in measurement accuracy. Therefore, the number of packets in a stream should be increased appropriately.

3.5.2. Effect of packet fragmentation

In the case, where TCP packets are transmitted through a queue or node for which the Maximum Transmission Unit (MTU) is smaller than the packet size, the packets will be fragmented into several pieces in the network. The problem here becomes a question of whether measurement result will still be accurate if the packets in measurement streams become fragmented somewhere on the way to the receiver. We argue that fragmentation has little effect on the measurement results. The measurement algorithm is based on the increasing trend of the packet stream in order to estimate available bandwidth. Even with fragmentation, the stream still shows an increasing trend when and only when the transmission rate is larger than the available bandwidth. However, fragmentation does increase the packet processing overhead, which may in turn raise the increasing trend of packet streams if it occurs at a bottleneck link. This may lead to a slight underestimation in the measurement results.

3.5.3. Effect of packet retransmission

When 3 dupACKs arrive and TCP packet retransmission occurs, the measurement program transmits the retransmitted packet then immediately releases all packets stored in the ImTCP Buffer. While the dupACKs arrive, the measurement program cannot determine the arrival intervals of the ACKs of the measurement streams, therefore, it cannot deliver measurement results. The program stops sending measurement streams and waits until a new ACK, instead of dupACKs, arrives. This is done to that the network has recovered from the congestion, and then measurements are restarted. Thus, packet retransmission only interrupts the measurements for a while.

4. Simulation results

4.1. Effect of parameters

4.1.1. N and the measurement accuracy

Fig. 7 shows the network model used in the ns-2 simulation. A sender host connects to a receiver host through a bottleneck link. The capacity of the bottleneck link is 100 Mbps and the one-way propagation delay is 90 ms. All of the links from the endhosts to the routers have a 100-Mbps bandwidth. There is cross traffic 1, 2 and 3 generated by endhosts connecting to the routers. The cross traffic is made up of UDP packet flows, in which various packet sizes are used according to the monitored results in the Internet reported in Ref. [17]. We make the available

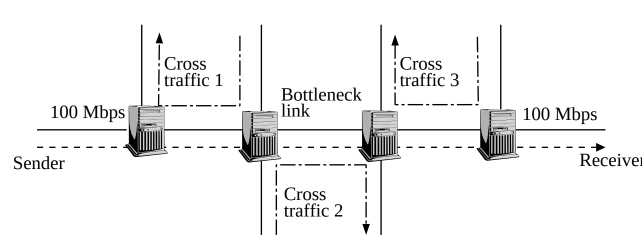
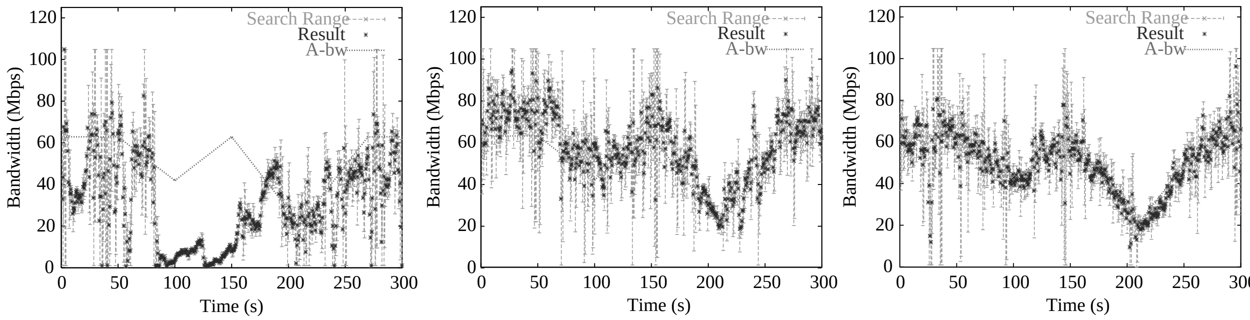


Fig. 7. Network model for evaluation of ImTCP.



(a) N=3 (b) N=5 (c) N=8

Fig. 8. Results of the proposed measurement algorithm.

bandwidth on the bottleneck link fluctuate by changing cross traffic 2's rate. Cross traffic 1 and 3 is for adding noise to the transmission delay of ACK packets.

To avoid counting on the effect from TCP behaviors, we investigate the results of the measurement algorithm when the sender uses the UDP streams for the measurement. In this case, the receiver simply echoes the UDP streams back to the sender. We show results in which we turn off cross traffic 1 and 3 and change the available bandwidth as follows: from 0 to 50 s, the available bandwidth is 60 Mbps; from 50 to 100 s, decreases to 40 Mbps; from 100 to 150 s, increases to 60 Mbps; from 150 to 210 s, decreases to 20 Mbps; from 120 to 270 s, increases to 60 Mbps; and from 270 to 300 s the available bandwidth is 60 Mbps. The simulation results are shown in Fig. 8. These figures indicate that when N is 3, the measurement results are far from the correct values. That is because, when N is very small, we cannot determine the increasing trend of the streams correctly in Step 3 in the proposed algorithm, which leads to the incorrect choice of sub-range in Step 4. When N becomes larger than five, on the other hand, the estimation result accuracy increases.

With a large value of N, packet storing time for one measurement stream becomes longer. Therefore, we want to keep the value as small as possible to avoid degrading the TCP transmission rate. We use $N=5$ as the default setting. In case the measurement accuracy is required, the N much be set to a larger value. In the following simulations, when there is no explicit mention, we use $N=5$.

4.1.2. Effect of setting of *m*

We next examine number of measurement results yielded in 80 (s) of simulation by a ImTCP connection when the available bandwidth is set to 3 Mbps. Table 1 shows the number of measurement results when N is set to different values. The results when using the proposed setting are shown in the column 'Pro' of the table. We vary the value of m to find at which value, the number of measurement results is almost the same as that in case $m=N$ (the underlined values). When $N=5$, $m=2$ is a good setting, because the number of results maintain highly while the average packet storing time is smallest. But when $N=6$, the optimal value of m changes to 4 and $m=2$ becomes a very bad setting because it decrees the number of results. Thus, the ideal value of m depends on the value of N. On the other hand, the dynamic setting always delivers large number of measurement results while the average packet waiting time is kept low.

4.1.3. Packet waiting time *T*

We next examine the number of measurement results of ImTCP when we set T to 0.04, 0.01, 0.004 s. Table 2 shows the values when we set available bandwidth to 4 and 7 Mbps. The 'Pro.' column shows the correspondent values when we use the proposed setting for T. When T takes small values such as 0.004 s or 0.01 s, ImTCP often fails to create measurement streams, therefore, the number of measurement results is small. On the other hand, as shown in Fig. 9, when T takes a large values, such as 0.04 s, the waiting time of the packets for stream creation is long. As a result, we found that the transmission rate of ImTCP when $T=0.04$ s is degraded. In contrast, the proposed setting for T can eliminate the cases when the packet waiting time is long, while maintaining the number of the measurement results.

4.2. Comparison with existing inline measurement methods

We set Cross traffic 1's transmission rate to 5 Mbps, Cross traffic 3's rate to 15 Mbps and changes Cross traffic 2' rate so that the available bandwidth fluctuates as shown by the line 'Available bandwidth' in Fig. 10. We show

Table 1
Number of measurement results

M	2	3	4	5	6	7	Pro.
$N=5$	387	441	424	389	–	–	424
$N=6$	200	216	406	392	370	–	394
$N=7$	77	173	277	279	348	349	359

Table 2
Number of measurement results

$T(s)$	0.04	Pro.	0.01	0.004
A-bw = 4 Mbps	379	371	324	105
A-bw = 7 Mbps	486	488	423	298

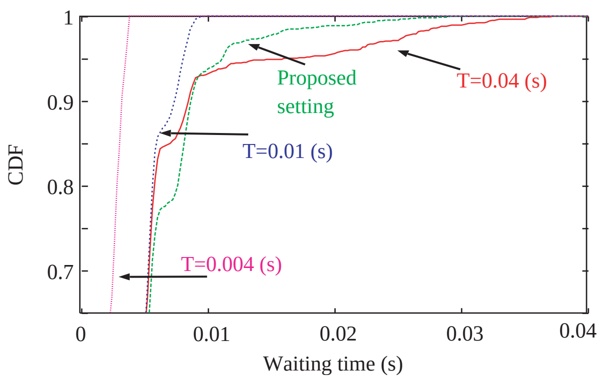


Fig. 9. CDF of the waiting time of the first packet in measurement streams.

the average measurement results of every 0.5 s of ImTCP, Westwood [14], the method proposed by Hoe [13] and TCPRab [15] in the network condition. In fact, the method by Hoe's performs only one measurement right after the connection starts. To compare with other methods, we repeat the measurements in every RTT.

Fig. 10(a) and (b) show that TCP-Rab can delivers accurate measurement results sometimes because the measurements do not interfered by the Cross traffic 1 and 3. Hoe's method is based on only three closely transmitted ACK packets so the affect from Cross traffic 1 and 3 is also small. Westwood perform worse in this condition because it counts on the arrival intervals of all ACK packets. However, the methods are all passive measurements so no one can detect the real value of available bandwidth if it changes fast from low to high. In contrast, ImTCP can detect the changes of available bandwidth fast because it actively adjusts the transmission rate of packets, even in the present of Cross traffic 1 and 3.

4.3. Effect of ImTCP on other traffic

To investigate the effect of inline measurement on other traffic sharing the network, we compare the case of ImTCP to that of Reno TCP using the network model depicted in Fig. 7 with the Cross traffic 1 and 3 turn off. Cross traffic 2 is changed to Web traffic involving a large number of active Web document accesses. We use a Pareto distribution for the Web object size distribution. We use 1.2 as the Pareto shape parameter with 12 kb as the average object size. The number of objects in a Web page is eight. The capacity of the bottleneck link is set to 50 Mbps. We use a large buffer (1000 packets) in the router at the shared link to help ImTCP/Reno TCP connections achieve high throughput because, here, the effect of ImTCP/Reno TCP connections on Web traffic is the focus of the simulation. We activate ImTCP and Reno TCP in turn in the network.

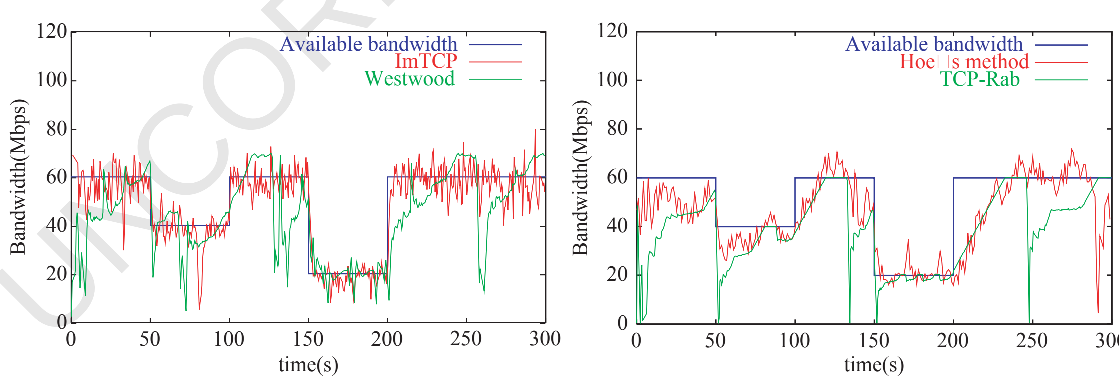
We run the simulation for 500 s and find that the average throughput of ImTCP is 25.2 Mbps while that of Reno TCP is 23.1 Mbps. The results therefore show that data transmission speed of ImTCP is almost the same as that of Reno TCP.

We compare the effect of ImTCP and Reno TCP on Web page download time in Fig. 11. This figure shows cumulative density functions (CDFs) of the Web page download time of Web clients. We can see that ImTCP and Reno TCP have almost the same effect on the download time of a Web page. This indicates that inline measurement does not affect other traffic sharing the link with ImTCP. Small graph in Fig. 11 also confirms that the ImTCP measurement result reflects the change in available bandwidth well.

4.4. Bandwidth utilization and fair share

Two important characteristics of the Internet transport protocol are full utilization of link bandwidth and fair sharing of bandwidth among connections. We use the following simulations to show that ImTCP has these two characteristics. We use the network topology shown in Fig. 12 with many ImTCP connections sharing a bottleneck link. Using a small buffer (200 packets) in the router at the bottleneck link to force conflict among connections, we vary the number of ImTCP connections while observing total throughput and fairness among the connections.

In Table 3 we show the Jain's fairness index [18] for the ImTCP connections as well as the total transmission rate of



(a) ImTCP and TCP Westwood

(b) method by Hoe and TCP-Rab

Fig. 10. Average measurement results of inline measurement methods.

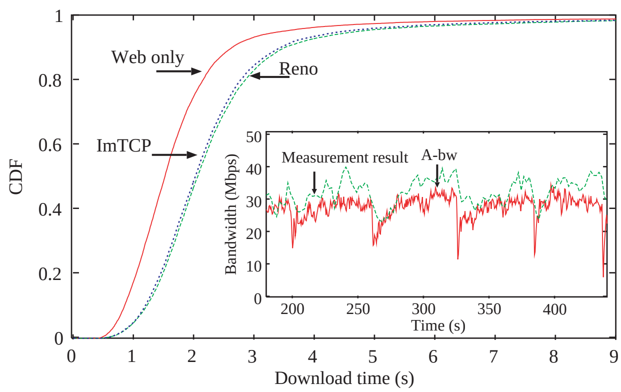


Fig. 11. Comparison of Web page download times.

ImTCP connections in Mbps when the capacity of the bottleneck link is set to 50, 60, and 70 Mbps. The number of connections is also varied. Also shown are the transmission rates when ImTCP is replaced by Reno TCP.

This Jain's fairness index takes a value from 0 to 1; a share is considered fair as its index is near 1. We can see that the ImTCP connections share the bandwidth link fairly because the index is always near to 1. Due to the small buffer size of the bottleneck link, when the number of connections are small the total throughput is not very high. When the number of connections is large, total throughput increases. We can see that ImTCP and Reno TCP have almost the same link utilization regardless of the number of connections.

4.5. TCP-friendliness and TCP-compatibility

ImTCP is *TCP*-friendly; it achieves the same throughput as Reno TCP under the same condition. Simulation results shown in Table 3 confirm this. Although ImTCP buffers packet stream at the sender host, the buffered packets is quickly transmitted after each transmission of a packet stream (in the EMPTY BUFFER state). Therefore, there is almost no degradation in transmission speed of data packets. A network protocol is called TCP-compatible if the connections using this protocol fairly share the bandwidth in a bottleneck link with Reno TCP [19]. We examine the TCP-compatibility of ImTCP by observing the throughput of ImTCP connections when they coexist with Reno TCP connections and non-TCP traffic. The non-TCP traffic is

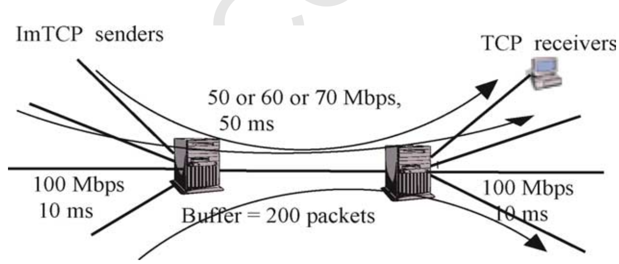


Fig. 12. Network model for investigating bandwidth utilization and fair share.

Table 3
Fairness and link utilization of ImTCP

Capacity	#flows	Jain's index	ImTCP	Reno
50	2	0.999	44.4	45.6
	10	0.997	46.7	46.0
	24	0.986	47.6	46.1
60	2	0.999	53.2	53.1
	10	0.992	55.3	54.0
	24	0.995	56.7	54.11
70	2	0.999	59.9	60.6
	10	0.992	63.7	61.9
	24	0.992	65.9	62.0

indicated by a 0.1 Mbps UDP flows with randomly varied packet size (300–600 bytes). All TCP and non-TCP traffic conflict at the 50 Mbps bottleneck link. We use the same number of ImTCP and Reno TCP connections.

The ratio of the total throughput of ImTCP connections to that of Reno TCP connections is shown in Fig. 13. When the ratio is around 1, ImTCP is TCP-compatible. The horizontal axis shows the total number of the TCP connections. In the current version of ImTCP, there is no time interval between two measurements. The result of this version is shown by the line numbered 0. We can see that ImTCP receives lower throughput than Reno TCP. The reason is as follows. Some of packets of ImTCP may not be transmitted in burst due to the affect of packets buffering at the sender. On the other hand, traditional TCP connections in competing environment have the trend to transmit packets in a bursty fashion. When the packets of ImTCP collide with the bursts of packets of Reno TCP, they have higher probability to be dropt. Therefore, ImTCP with high measurement frequency may lost more packets when conflicting with Reno TCP, leading to a lower throughput.

The simple and effective way to overcome this problem is increasing the measurement interval of ImTCP. We next consider the cases when the measurement intervals are 12, 15 and 20 RTTs, and show the results by the line numbered 12, 15 and 20, respectively, in Fig. 13. Note that the RTT in this case is 0.14 s and each measurement takes at most 4 RTTs. Therefore, 12, 15 and 20 RTT interval means ImTCP

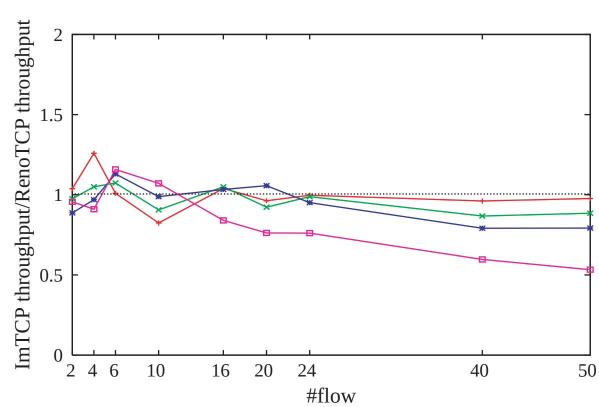


Fig. 13. Comparison of ImTCP and Reno TCP throughput. The number on the lines are the number of RTT between two measurements of ImTCP.

releases measurement results in 2.24(s), 2.66(s) and 3.36(s), respectively. When the measurement interval is relatively small, ImTCP achieves lower throughput than Reno TCP. On the other hand, when the measurement interval is equal to or larger than 20 RTTs, ImTCP is compatible to Reno TCP. In other words, when the measurement frequency is smaller than a certain value (in this simulation, that is 1/3.36 times per second) there is a trade-off relationship between the TCP compatibility and the measurement frequency.

In such a heavy congested network that there is no available bandwidth even when ImTCP does not exist, ImTCP must be TCP-compatible in order to gain the equal throughput to other connections. Moreover, in this environment, the measurement results themselves usually do not bring so much valuable information so they will be not required updated frequently. Therefore, in this case, ImTCP must take a low measurement frequency. When the network is vacant, ImTCP will not conflict with other connections so much. In this case, TCP-compatibility does not strictly required, because ImTCP is TCP-friendly so that ImTCP will perform exactly like traditional TCP. Besides, the information about the vacancy in the network will be of interest. In this case, ImTCP should increase its measurement frequency. Thus, there should be a dynamic adjustment for the measurement frequency according to the network status. We will consider the problem in our future works.

5. Transmission modes of ImTCP

5.1. Background transmission

The transmission for backup data or cached data (background traffic) should not degrade throughput of other traffic (foreground traffic), which may be more important. We introduce an example showing that ImTCP successfully uses the results of bandwidth availability measurements to prevent its own traffic from degrading the throughput of other traffic. We call this type of ImTCP data transmission background mode. The main idea is to set an upper bound on the congestion window size according to estimated values so that the transmission rate does not exceed the available bandwidth. This reduces the effect ImTCP has on other traffic in the same network links. We use the following control mechanism. When

$$g\mathrm{RTT}A > mN$$

we set

$$\mathrm{MaxCwnd} = g\mathrm{RTT}A$$

where A is the estimated value of available bandwidth, MaxCwnd is the upper bound of the congestion window size and N is the number of packets for a measurement stream. The parameter g can range from 0 to 1. When g is small, ImTCP uses less bandwidth and interferes only very slightly with foreground traffic. When g is near 1, ImTCP uses more bandwidth and its effect on foreground traffic grows. We set the upper bound of the congestion window size (MaxCwnd) to $g\mathrm{RTT}A$ only when the value is large enough for ImTCP to continue performing measurements well.

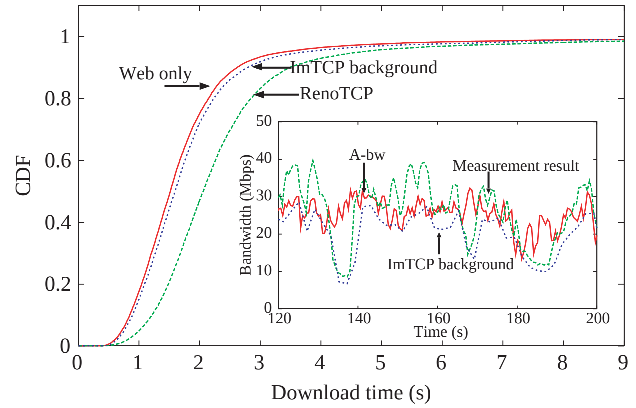


Fig. 14. Average of Web page download time.

We examine the behavior of ImTCP in background mode when foreground traffic is originated with Web document transfers. We replace the ImTCP connection in the simulation in Section 4.3 with a background mode ImTCP connection. Fig. 14 compare the download time for Web pages under ImTCP and Reno TCP. We find that ImTCP has only a very small effect on the download time of the foregroundWeb traffic. The average throughput of ImTCP in this case is about 72% that of Reno TCP. The small graph in Fig. 14 shows the measurement value and throughput of ImTCP connection as a function of simulation time in this case. Note that the throughput of ImTCP does not approach the actual value of available bandwidth. This indicates that ImTCP background mode is successfully avoiding interference with Web traffic.

5.2. Full-speed transmission

We introduce another example of a modified congestion control mechanism to show that ImTCP can enhance link utilization using its measurement results. We explain the study in details in [20].

To improve TCP throughput in wireless or high-speed networks, we introduce an available-bandwidth-aware window size adjustment. The idea is to use the measurement result to adjust the increasing speed of the congestion window size. When the available bandwidth is large, the window size increases quickly to make full use of available bandwidth, and when the available bandwidth is small due to the existence of other traffic, the window size increases slowly. We call this type of ImTCP data transmission full-speed mode.

In the congestion avoidance phase, we do not increase the congestion window size (Cwnd) by one in every RTT.

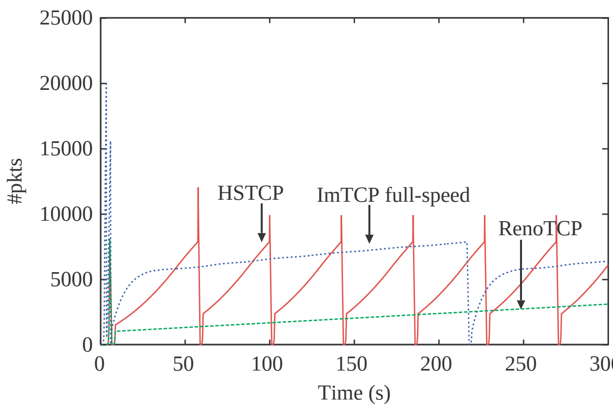


Fig. 15. Comparison of TCP window sizes.

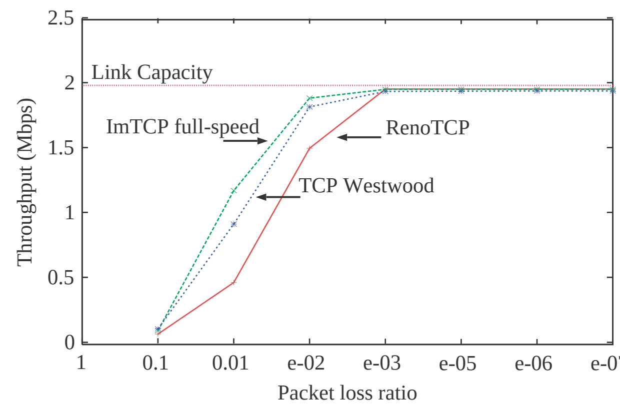


Fig. 16. TCP throughput in wireless network.

Instead, we use the following adjustment.

$$\mathrm{Cwnd} \leftarrow \mathrm{Cwnd} + \max\left(1, h\left(1 - \frac{\mathrm{Cwnd}}{V}\right)\right)$$

$$V = ARTT$$

In the equation, $h (h \geq 1)$ is a parameter that determines how fast the window size increases. If h is large, ImTCP can successfully utilize the bandwidth link. When h is small or equal to one, ImTCP behaves the same as Reno TCP.

We perform the following simulation to investigate the performance of ImTCP in full-speed mode. The ImTCP sender and ImTCP receiver is connected by two routers with Gigabit links. The 500 Mbps link between the two routers becomes the bottleneck link in the path. We assume the buffer of the TCP receiver is large so the TCP throughput can achieve 500 Mbps.

Fig. 15 shows the changes in the window size of ImTCP in full-speed mode, High-Speed TCP (HSTCP) [21] and Reno TCP in the network. Reno TCP requires a long time to reach a large window size. HSTCP increases the window size quickly to fully use the free bandwidth, however, the increasing speed is non-sensitive to the available bandwidth such that packet loss events occur frequently. Therefore, overall, the throughput of HSTCP is not as large as expected. ImTCP increases the window size quickly when the window size is small and decreases the speed when its transmission rate reaches the available bandwidth to avoid packet losses. Therefore, the throughput of ImTCP is better than the others.

Finally, we compare the throughput of ImTCP in full-speed mode with Reno TCP in a wireless network. We insert a 2 Mbps network link in the path between a TCP sender and TCP receiver to simulate a wireless link. We vary the packet loss rate of the network links and find that ImTCP can achieve a larger throughput than TCP Westwood and Reno TCP when the loss rate is high, as shown in Fig. 16.

Parameter h is set to 100 in this case. When the packet loss rate is high, a higher value for parameter h can help ImTCP obtain higher available bandwidth. When the packet loss rate is low, the value of h should be low so that ImTCP will share bandwidth fairly with other traffic.

6. Conclusions

In this paper, we introduced a method for measuring the available bandwidth in a network path between two end hosts using an active TCP connection. We first constructed a new measurement algorithm that uses a relatively small number of probe packets yet provides periodic measurement results quickly. We then applied the proposed algorithm to an active TCP connection and introduced ImTCP, a version of TCP that can measure the available bandwidth. We evaluated ImTCP through simulation experiments and found that the proposed measurement algorithm works well with no degradation of TCP data transmission speed. We also introduced examples of ImTCP special transmission modes.

In future projects, we will make ImTCP to be completely TCP-compatible without decreasing measurement frequency requires. We will also develop new transmission modes for ImTCP as well as evaluate the performance of the modes introduced in the paper.

References

[1] R. Anjali, C. Scoglio, L. Chen, I. Akyildiz, G. Uhl, ABEst: an available bandwidth estimator within an autonomous system Proceedings of IEEE GLOBECOM 2002.

[2] S. Seshan, M. Stemm, R.H. Katabi, SPAND: Shared passive network performance discovery, in: Proceedings of the 1st Usenix Symposium on Internet Technologies and Systems (USITS '97), 1997, pp. 135–146.

[3] R.L. Carter, M.E. Crovella, Measuring bottleneck link speed in packet-switched networks, Tech. Rep. TR-96-006, Boston University Computer Science Department (Mar. 1996).

[4] M. Jain, C. Dovrolis, End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput Proceedings of ACM SIGCOMM 2002.

[5] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, L. Cottrell, PathChirp: efficient available bandwidth estimation for network paths Proceedings of Passive and Active Measurement 2003.

[6] N. Hu and P. Steenkiste, Evaluation and characterization of available bandwidth probing techniques, IEEE Journal on Selected Areas in Communications 21 (6).

[7] J. Strauss, D. Katabi, F. Kaashoek, A measurement study of available bandwidth estimation tools Proceedings of Internet Measurement Conference 2003.

[8] D. Andersen, H. Balakrishnan, M. Kaashoek, R. Morris, Resilent overlay networks Proceedings of ACM SOSPs 2002.

[9] The Internet Bandwidth Tester (TPTEST), available at http://tptest.sourceforge.net/about.php.

[10] M. Gerla, Y. Sanadidi, R. Wang, A. Zanella, C. Casetti and S. Mascolo, TCP Vegas: New techniques for congestion detection and avoidance, in: Proceedings of the SIGCOMM'94 Symposium, 1994, pp. 24–35.

[11] S. Savage, Savage, Sting: a TCP-based network measurement tool Proceedings of USITS '99 1999.

[12] Sprobe, available at http://sprobe.cs.washington.edu.

[13] J.C. Hoe, J.C. Hoe, Improving the start-up behavior of a congestion control scheme for TCP, Proceedings of the ACM SIGCOMM Conference on Applications Technologies, Architectures, and Protocols for Computer Communications, vol. 26, 4, ACM Press, New York, 1996. pp. 270–280 (URL citeseer.nj.nec.com/hoe96improving.html.).

[14] M. Gerla, B. Ng, M. Sanadidi, M. Valla, R. Wang, TCP Westwood with adaptive bandwidth estimation to improve efficiency/friendliness tradeoffs, To appear in Computer Communication Journal.

[15] T. Xu-hong, L. Zheng-lan, Z. Miao-liang, TCP-Rab: a receiver advertisement based TCP protocol, Journal of Zhejiang University Science 5 (11) (2004) 1352–1360.

[16] R. Stevens, TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 1994.

[17] NLANR web site, available at http://moat.nlanr.net/Datacube/.

[18] R. Jain, Jain, The art of computer systems performance analysis: techniques for experimental design Measurement, Simulation, and Modeling, Wiley-Interscience, New York, 1991.

[19] S. Jin, L. Guo, I. Matta and A. Bestavros, TCP-friendly SIMD congestion control and its convergence behavior, in: Proceedings of ICNP'01: The 9th IEEE International Conference on Network Protocols, 2001.

[20] T. Iguchi, G. Hasegawa, M. Murata, A new congestion control mechanism of tcp with inline network measurement Proceedings of ICOIN 2005, (Jeju) 2005 pp. 109–121.

[21] S. Floyd, Highspeed TCP for large congestion windows, RFC 3649.

Cao Le Thanh Man received the MS degree from Graduate School of Information Science and Technology, Osaka University, Japan, in 2004. He is now a doctoral student at the same school. His research interests include network performance measurement and evaluation, TCP protocol design and evaluation. He is a student member of IEICE.

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

Masayuki MURATA received the ME and DE degrees in Information and Computer Sciences from Osaka University, Japan, in 1984 and 1988, respectively. In April 1984, he joined Tokyo Research Laboratory, IBM Japan, as a Researcher. From September 1987 to January 1989, he was an Assistant Professor with Computation Center, Osaka University. In February 1989, he moved to the Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University. From 1992 to 1999, he was an Associate Professor in the Graduate School of Engineering Science, Osaka University, and from April 1999, he has been a Professor of Osaka University. He moved to Advanced Networked Environment Division, Cybermedia Center, Osaka University in 2000, and moved to Graduate School of Information Science and Technology, Osaka University in April 2004. He has more than two hundred papers of international and domestic journals and conferences. His research interests include computer communication networks, performance modeling and evaluation. He is a member of IEEE, ACM, The Internet Society, IEICE and IPSJ.