Available online at www.sciencedirect.com
SCIENCE DIRECT
Computer Communications 28 (2005) 819–828
computer communications
www.elsevier.com/locate/comcom

One-way delay estimation and its application

Jin-Hee Choi*, Chuck Yoo

Department of Computer Science and Engineering, Korea University, Asan science building 236, 136701 Seoul, South Korea

Received 9 April 2004; revised 3 November 2004; accepted 24 November 2004
Available online 15 December 2004

Abstract

Delay estimation is a difficult problem in computer networks. Accurate one-way delay estimation is crucial because it serves a very important role in network performance and thus application design. RTT (Round Trip Time) is often used as an approximation of the delay, but because it is a sum of the forward and reverse delays, the actual one-way delay cannot be estimated accurately from RTT.

To estimate one-way delay accurately, this paper proposes a new scheme that analytically derives one-way delay, forward and reverse delay, respectively. We show that the performance of TCP can improve dramatically in asymmetric networks using our scheme. A key contribution of this paper is that our one-way delay estimation is much more accurate than RTT estimation so that TCP can quickly find the network capacity in the slow start phase. Since RTT is the sum of the forward and reverse delays, our scheme can be applied to any protocol that is based on RTT.

Keywords: Delay measurement; End-to-end delay; Clock offset; Round trip time

1. Introduction

We address a fundamental problem in computer networks, namely, how to estimate one-way delay time between the sender and the receiver. It would be an easy job to get the one-way delay if there is global time synchronization between the sender and the receiver. If there is, the forward delay can be simply calculated by taking the difference of receiver's clock and sender's timestamp. Receiver can write this value into the ACK packet's header, and the reverse delay is also obtained similarly. Unfortunately, one-way delay cannot be obtained accurately because the clocks at end systems are not synchronized with each other. Even with a considerable amount of research efforts [1–3], the clock synchronization problem is not solved to the level suitable for calibrating packet's forward and reverses delay [4–6]. Furthermore, in heterogeneous and massive networks such as Internet, it is even more difficult to guarantee the synchronized clock.

RTT is often used as an approximation of the delay. RTT is a key parameter in the end-to-end communication, which is used for determining timeout and deciding transmission rate at the sender. A basic assumption behind RTT is that the underlining network is symmetric. In other words, each of forward delay and reverse delay is roughly 1/2 of RTT [7]. However, in today's networks such as cable modem networks, direct broadcast satellite, asymmetric digital subscriber loop (ADSL) and a variety of 3G wireless data networks, the bandwidth in the forward direction is often larger by orders of magnitude than that of the reverse link [8–12]. Furthermore, a large-scale study of Internet routing has found that paths through the Internet are often asymmetric [13,14], meaning that the routers visited in the forward and the reverse directions often differ. Also, even if the packets go through the same route, and of the same size, the packets experience different levels of queuing inside the network. Consequently, all of those various properties make RTT a poor approximation of one-way delay.

If the forward delay is off too much from 1/2 of RTT, many protocols based on RTT end up with severely misusing the network resources. For example, when a reverse path is congested, the reverse delay gets much larger than the forward delay. When protocols like TCP use RTT, because RTT is the sum of the two, the protocols estimates the available bandwidth inaccurately due to longer forward delay than what it should be. Also, the retransmission timeout value (RTO) becomes inaccurate due to the same reason.

* Corresponding author. Tel.: +82 232903639; fax: +82 9226341.
E-mail address: jhchoi@os.korea.ac.kr (J.-H. Choi).

0140-3664/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2004.11.010

This paper proposes an accurate delay estimation scheme that analytically calculates the forward and reverse delay, respectively, and we apply the new scheme to TCP in the asymmetric networks, in order to show that TCP start-up behavior based on accurate forward delay leads to much more efficient network utilization than that with RTT. The rest of the paper is organized as follows. In Section 2, we present a shape of typical one-way delay that motivated us to design a new estimation scheme. Section 3 describes our delay estimation scheme. Section 4 explains the implementation protocol for using our scheme and the implementation details. In Section 5, the impact of our scheme on the network traffic is analyzed, and Section 6 provides an application example that applies our scheme to the asymmetric networks. In Section 7, we show the simulation results and analyze the reason, and we discuss the results and an unsolved problem of our scheme in Section 8. Finally, Section 9 concludes the paper.

2. Motivation and background

2.1. Motivation

Before we introduce our estimation scheme, let us first examine a result that shows the difference of 'accurate' forward delay and round trip time at TCP sender. This result is from the simulation in Section 7, and it is illustrated in Fig. 1.

In this graph, *X*-axis is simulation time, and *Y*-axis is the one-way delay and the RTT/2. The forward delay is calculated by subtracting the sender's timestamp in each packet header from the receiver's clock, under the assumption of clock synchronization. Fig. 1 shows that there can be the large difference of the forward delay and the RTT/2 even in symmetric network, which consequently affects the protocol performance negatively because RTT is used as an indicator about network condition in the end system. Obviously, the estimation of one-way delay, not using RTT, is the solution of this problem.

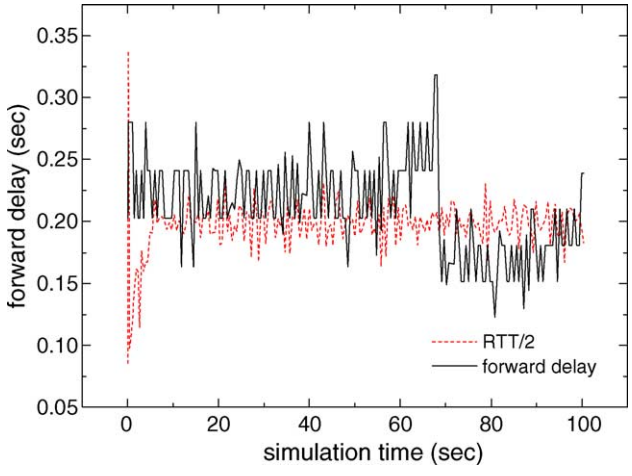

Fig. 1. Forward delay and RTT/2.

The attempts in previous work to estimate one-way delay are categorized in roughly two. An approach is to use the difference of the sender timestamp and receiver timestamp after synchronizing two clocks. Many schemes attempted to use physical clock synchronization algorithm, but it is known that clock synchronization is difficult to be applied to massive distributed computing environment. The other is to do calibration by taking the clock skew in the two timestamps into account. Paxson [1] and Moon [2] addressed this problem and proposed a solution, respectively. However, these schemes cannot be practically applied to the transport protocols that have to adjust their behavior depending on the network condition because they require the calibration time to obtain the one-way delay form the measurement.

Our new delay estimation scheme has the following characteristics:

- It does not assume time synchronization between the sender and the receiver.
- It tracks the variations of forward and reverse delay accurately even when the delay value changes dynamically due to the traffic pattern and congestion in the network.
- It calculates the delay in a short time, so the value can be used in many protocols that adjust their behavior depending on the network condition.
- It is very simple so it can be implemented practically.

2.2. Basic clock terminology

Let us define basic terminology for discussing the characteristics of the clocks used in following study. Mills [6] defines a nomenclature for describing clock characteristics, which we will use as appropriate.

- *Frequency*. The rate at which the clock progresses.
- *Offset*. The difference between the time reported by the clock and the 'true' time as defined by national standards. The 'true' time is reported by 'true' clock at any moment, and runs at a constant rate. For example, the difference of the sender and the 'true' clock is the offset ΔC_s, and the difference of the sender and receiver clock is the relative offset $\Delta C_{s,r}$.
- *Skew*. The frequency difference between the clock and national standards. That is, the instantaneous difference between the readings of any two clocks is called their skew.

3. Delay derivation

Our scheme uses one packet per RTT to estimate the delay. When multiple packets are sent in RTT, our scheme marks one packet to avoid confusion, and the ACK of the marked packet is sent back. Fig. 2 shows a typical interchange of packets and ACKs.

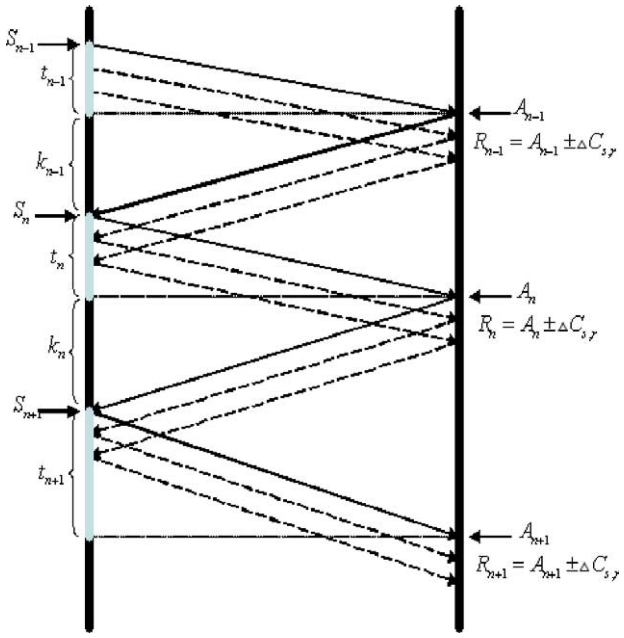

Fig. 2. Typical packet exchange of TCP sender and receiver.

Let us introduce the notations for clocks, timestamp and delays used in our derivation.

- C_s: sender clock.
- C_r: receiver clock.
- S_n: the transmission time of the sender for nth packet according to C_s.
- R_n: the arrival time of nth packet at the receiver according to C_r.
- A_n: the arrival time of nth packet at the receiver according to C_s.
- t_n: the forward delay of the nth packet according to C_s; $A_n - S_n$.
- k_n: the reverse delay of the nth packet according to C_s; $S_{n+1} - A_n$.
- RTT(s,n): the round trip time of $(n-1)$ th packet at the sender according to C_s.
- RTT(r,n): the round trip time of ACK for $(n-1)$ th packet at the receiver according to C_r.
- $\Delta C_{s,r}$: the relative offset of a clock C_s at sender with respect to a clock C_r at receiver.

It is very important to get the accurate A_n because both forward and reverse delays are calculated based on A_n, and the relative offset $\Delta C_{s,r}$ can be estimated by obtaining accurate A_n. However, A_n is not known at the receiver, and it is the reason that clock synchronization is difficult in distributed systems. Therefore, our scheme does not use accurate spot of A_n but utilizes the time length is same independent of the clock skew.

We begin our derivation by defining the meaning of RTT more clearly.

- RTT measured by sender: RTT(s,i)

Sender transmits a packet and measures the time upon receiving ACK, and the time difference is RTT. Therefore, it denotes RTT $(s, n+1) = t_n + k_n$, and it is a measured RTT by the sender, and it is a measured RTT at S_{n+1} according to C_s by the sender. Therefore, we get the following equation, Eq. (1)

$$\mathrm{RTT}(s, n+1) = S_{n+1} - S_n = t_n + k_n \tag{1}$$

- RTT measured by receiver: RTT(r,i)

Receiver measures RTT by the difference of the time of sending the previous ACK and the current ACK. So, it denotes $\mathrm{RTT}(r,n) = R_n - R_{n-1}$, and it is a measured RTT at R_n according to C_r by the receiver. Also, we get the relation of $R_{n+1} - R_n = A_{n+1} - A_n$ since $R_n = A_n \pm \Delta C_{s,r}$ and $R_{n+1} = A_{n+1} \pm \Delta C_{s,r}$ are allowed. From this relation, we can know that it is possible to get the forward delay consistent with C_s even through we use RTT calibrated according to C_r by receiver (we assume that the relative offset is constant on adjacent RTT phases). Thus, we get another equation, Eq. (2)

$$\mathrm{RTT}(r,n) = R_n - R_{n-1} = A_n - A_{n-1} = t_n + k_{n-1} \tag{2}$$

Since the RTTs measured at the sender and receiver have forward delay t_n as common, subtracting the two equations results in:

$$\mathrm{RTT}(s, n+1) - \mathrm{RTT}(r,n) = k_n - k_{n-1} \tag{3}$$

By summing the difference from 1 to n, we get the following expression

$$\begin{aligned} \mathrm{RTT}(s,2) - \mathrm{RTT}(r,1) &= k_1 - k_0 z \\ \mathrm{RTT}(s,3) - \mathrm{RTT}(r,2) &= k_2 - k_1 \\ \mathrm{RTT}(s,4) - \mathrm{RTT}(r,3) &= k_3 - k_2 \\ &\cdots \\ \mathrm{RTT}(s,n+1) - \mathrm{RTT}(r,n) &= k_n - k_{n-1} \\ \sum_{i=1}^{n} \mathrm{RTT}(s,i+1) - \sum_{i=1}^{n} \mathrm{RTT}(r,i) &= k_n - k_0 \end{aligned} \tag{4}$$

When n is 0, Eq. (1), $\mathrm{RTT}(s, n+1) = t_n + k_n$ becomes $\mathrm{RTT}(s,1) = t_0 + k_0$.

Now, we can rewrite above equations as rearranging k_n and k_0, respectively

$$k_n = \mathrm{RTT}(s, n+1) - t_n, \quad k_0 = \mathrm{RTT}(s,1) - t_0$$

Thus, we get

$$\begin{aligned} k_n - k_0 &= \sum_{i=1}^{n} \mathrm{RTT}(s, i+1) - \sum_{i=1}^{n} \mathrm{RTT}(r,i) \\ &= \mathrm{RTT}(s, n+1) - t_n - \mathrm{RTT}(s,1) + t_0 \end{aligned} \tag{5}$$

Finally, we have forward delay t_n

$$t_n = t_0 - \sum_{i=1}^{n}[\mathrm{RTT}(s,i) - \mathrm{RTT}(r,i)] \quad (6)$$

Also, we have reverse delay, k_n

$$k_n = -t_0 + \sum_{i=1}^{n+1}\mathrm{RTT}(s,i) - \sum_{i=1}^{n}\mathrm{RTT}(r,i) \quad (7)$$

Eqs. (6) and (7) mean that the forward and reverse delay could be calculated by sender-measured RTTs and receiver-measured RTTs. Note that the variation of network condition only depends on the difference of the sender-measured and receiver-measured RTT. Also, the equation implies that the variation of one-way delay could be tracked accurately if the receiver measures RTT, and returns this value to the sender.

4. Protocol implementation

To implement Eq. (6), we need to calculate three values: sender-measured RTT, receiver-measured RTT, and initial fixed-point value (t_0). We explain how to calculate each one below. The pseudo-code of our protocol is described in Fig. 3.

- RTT(s,i): all existing versions of TCP periodically measure RTT to calculate RTO or the sending rate. Therefore, it is easy for the sender to measure RTT.
- RTT(r,i): the most intuitive way of measuring RTT at the receiver is to measure the interval of ACKs. The receiver has to write such RTT values into the header of ACK packet since the sender needs the difference of the sender-measured and receiver-measured RTT in Eq. (6). An alternative is to use a TCP header option, but we think that it is possible to use unused field in the header of ACK packet.
- t_0: although Eq. (6) shows that the one-way delay can be analytically derived, the accuracy of the delay estimation is determined by t_0. Thus, a question is how to get proper t_0. When a TCP session is created, if there is nearly no congestion in the networks, using RTT/2 as t_0 would be reasonable. However, it is difficult to predict the network condition when TCP sessions are created. For that reason, we need a heuristic to reduce the error range of t_0

Sender's protocol

```
1. ON sending segment for RTT measurment:
2.     tcph.timestamp = current_time;
3.     probe_sequence++;
4. ON receiving new ACK:
5.     RTT_S = current_time − ack.timestamp;
6.     IF (ack.rtt_r < 0)
7.         exit;
8.     RTT_R = ack.rtt_r;
9.     IF (probingPhase == true)
10.        calculate average d;
11.    ELSE
12.        IF (firstDataACK == true)
13.                make t0;
14.        ELSE
15.            fd_diff = fd_diff+
16.                    (fd_prev_rtt_s − ack.rtt_r);
17.            fd_prev_rtt_s = RTT_S;
18.        END
19.        tn = t0 − fd_diff;
20.    END
```

Receiver's protocol

```
1. ON receiving TCP segment:
2.     IF (firstPacket == true)
3.         fd_prev_ack = current_time;
4.         ack.rtt_r = −1;
5.     ELSE
6.         ack.rtt_r = current_time − fd_prev_ack;
7.         fd_prev_ack = current_time;
8.     END
```

Fig. 3. Protocol for estimating forward delay.

up to the level of acceptance in the end-to-end protocol. First, we get upper bounds of t_0.

With a variable d as the difference of the forward and reverse delay ($t_0 - k_0 = d$), there are three cases of upper bounds:

Case 1: $d > 0$

We get $2t_0 > \mathrm{RTT}(s,0) > 2k_0$ by $t_0 > k_0$. By summing t_1 to $\mathrm{RTT}(s, 0) > 2k_0$ on both sides, we also get $t_1 + \mathrm{RTT}(s,0) > t_1 + 2k_0$. Now, we can rewrite new range of t_0 as follows by using expressions, $t_1 + k_0 = \mathrm{RTT}(r,0)$ and $t_1 = t_0 + \mathrm{RTT}(s,1) - \mathrm{RTT}(r,1)$

$$\frac{\mathrm{RTT}(s,1) - \mathrm{RTT}(r,1) + \mathrm{RTT}(r,0)}{2} < t_0 < \mathrm{RTT}(s,0) \quad (8)$$

Case 2: $d < 0$

With same process of the case $d > 0$, we can get new range as follows:

$$\frac{\mathrm{RTT}(s,1) - \mathrm{RTT}(r,1) + \mathrm{RTT}(r,0)}{2} > t_0 > 0 \quad (9)$$

Case 3: $d = 0$

Because t_0 and k_0 are same

$$t_0 = k_0 = \frac{\mathrm{RTT}(s,0)}{2} \quad (10)$$

To our best knowledge, however, there is no practical method to know the sign of d. Therefore, we choose a heuristic to predict it.

This heuristic method uses the property of one-way delay in Internet, which is that queuing delay tends to change steadily and is a dominant part in the whole delay. So we expect that d would not be highly different from the average difference of a few sample forward and reverse delays during probing phase. In other words, if the ratio of forward delay and RTT is larger than α, we choose $\mathrm{RTT}(s,1) - \mathrm{RTT}(r,1) + \mathrm{RTT}(r,0) + 2\mathrm{RTT}(s,0)/4$ as t_0 (the range average in case of $d > 0$). The reverse, if the ratio is smaller than β, we choose $\mathrm{RTT}(s,1) - \mathrm{RTT}(r,1) + \mathrm{RTT}(r,0)/4$ as t_0 (the range average in case of $d < 0$). Also, if the ratio is larger than β and smaller than α, we choose $\mathrm{RTT}(s,0)/2$ as t_0. α and β are approximation to middle values in 0, $\mathrm{RTT}(s,0)/2$ and $\mathrm{RTT}(s,0)$. Thus, we use 0.7 and 0.3 as α and β in our implementation and could confirm that these values work well by a lot of simulation experiments.

Note that the estimation cannot start until the segment and ACK are interchanged more than twice, that is, until the first receiver-measured RTT arrives.

5. Analysis of the impact of our scheme on the network traffic

Here we analyze the overhead of our scheme—additional packets to obtain RTT(*s,i*), RTT(*r,i*), and a small space in ACK header for RTT(*r,i*). The overhead differs in two cases: when our scheme is used with a feedback-based protocol like TCP, and when the scheme is used independently for pure measurement.

5.1. Our scheme with TCP

Because, in order to calculate RTO, TCP sender continues to measure RTT, our estimation scheme does not have to send any additional packets for RTT(*s,i*). TCP receiver also does not require any new packet, for the receiver can obtain RTT(*r,i*) by calculating the interval of ACKs. Usually, TCP receiver sends several ACKs every RTT since the sender transmits the data packets with the size of *cwnd* by its sliding window mechanism. It may confuse the calculation of the ACKs interval. An intuitive method to remove such ambiguity is that the TCP sender marks in the packet header whenever the sender transmits the packet for RTT(*s,i*) measurement. The receiver is able to measure the interval of ACKs more accurately by considering only the ACKs of the marked packets.

Therefore, our scheme does not add any overhead traffic with TCP. This merit is still valid even if the receiver uses the delayed ACK policy. The policy sends just an ACK on receiving generally two or more packets since normally the minimum *cwnd* is over two. Even with the delayed ACK policy, at least one ACK is generated every RTT. For this reason, we do not need to make any additional ACKs for RTT(*r,i*). So, our estimation scheme has a negligibly small overhead in the practical point of view even though the method requires additional header space to put the mark for removing the ambiguity in RTT(*s,i*) and RTT(*r,i*).

5.2. When our scheme is used independently

To get the RTTs at both sides, we need a simple ping-pong protocol that exchanges dummy control packet and its acknowledgement packet. At this case, all additional packets are the overhead traffic in the networks, but the overhead is not large since the control packet does not require any payload. Also, if the size of forward and reverse packet is same, the scheme can make more accurate estimation.

With assumption that the size of the packets is 40 bytes, and measurement run time is t, we can approximate the overhead network traffic as follows

$$\text{Overhead Traffic} = \frac{t}{\text{average(RTT)}} \times 40 \text{ bytes} \times 2 \quad (11)$$

This means that additional traffic of about 40 Kbytes is used in one-way delay estimation when a network has 200 ms average RTT for 100 s measurement.

```
Sender's protocol
1. ON receiving new ACK:
2.     IF ((slow_start == true) and
3.             (fd_estimation_available == true))
4.         fd = fd_estimation();
5.         IF (cwnd_timer_run == true)
6.             cwnd_timer_cancel();
7.             cwnd++;
8.         END
9.             cwnd_timer->timeout = fd × 2;
10.    END

1. ON expiring cwnd_timer:
2.       cwnd++;
```

Fig. 4. TCP fast increase.

6. Application to TCP

In this section, we explain how to apply our one-way delay estimation scheme to TCP in an asymmetric environment[1]. TCP shows severe performance drop in asymmetric networks and the reason is that the protocol is heavily dependent on feedback in the form of ACK. The variation in the ACK arrival rate, caused due to the asymmetry, makes the throughput degraded. To our knowledge, the realistic prevention scheme of TCP performance degradation in asymmetric networks is not proposed yet. We compare our scheme with what already have been proposed to show why our scheme is more effective and practical.

An intuitive solution to the asymmetry is controlling the congestion in ACK flow or reducing the number of ACK packets through filtering. Balakrishnan et al. [12] proposed ACK Congestion Control (ACC) and ACK Filtering (AF) to solve the performance reduction problem in the networks. Also, delayed ACK policy may be helpful for reducing the number of ACK in the end-to-end context. Those ACK control schemes alleviate the problem of congestion on the reverse bottleneck link by decreasing the frequency of ACKs, with each ACK potentially acknowledging several data packets. But these methods lead to have common problems, which are sender burstiness, slow *cwnd* growth in the slow start phase, and a decrease in the effectiveness of the fast retransmission mechanism [12].

Sender Adaptation' key idea is that the sender sends data in sizes of maxburst, even if the *cwnd* allows the transmission of more data. Also, the bursts are spaced by a timer, whose expiration value is the ratio of maxburst to *cwnd*/RTT. Therefore, large bursts of data get broken up into smaller chunks spread out over time. However, in this scheme, TCP is still dependent on the ACK arrival rate for the congestion window growth. As a result, this leads to under-utilize the forward path's bandwidth. If we can obtain the forward delay, TCP can increase the *cwnd* per ($2\times$ *forwarddelay*), at the least in the slow start phase[2]. This scheme achieves (1) faster growth of *cwnd* and (2) more efficient network resource utilization.

Such a basic concept was already proposed in Raisinhani et al. [8]. But they use a heuristic factor, called MAF, based on the network condition, to estimate ($2\times$*forwarddelay*). Thus their approach showed the possibility through the simulation but could not be used practically.

TCP's misunderstanding in the asymmetric networks is mainly due to inaccurate RTT. There often exists quite a time gap between data flow and ACK flow, while RTT is a sum of both the forward and reverse delay. Therefore TCP's adaptation based on RTT can misbehave in the asymmetric environments. If we estimate one-way delay with reasonable degree, TCP can reduce the effect of the variation in ACK arrival rates.

Our solution's key idea is letting the bandwidth estimation in the slow start phase be independent on the ACK arrival. In other words, TCP increases *cwnd* per ($2\times$*forwarddelay*) regardless of the ACK arrival in the slow start phase (this let TCP search the network capacity fast). Hence, TCP sends data packets like the TCP original mechanism of the ACK clocking. Note that our mechanism performs only when the asymmetry of one-way delay is severe, to reduce the impact of the unlikely error in our one-way delay estimation scheme on the other TCP sessions in the networks. That is, Fast Increase (FI) is not activated if the sender continues to receive ACKs before the *cwndtimer* is expired. This simple idea's pseudo code is described in Fig. 4. Through extensive simulation

[1] A network is said to exhibit network asymmetry with respect with TCP performance, if the throughput achieved is not solely a function of the link and traffic characteristics of the forward direction, but depends significantly on those of the reverse direction as well [12].

[2] In the slow start phase, TCP estimates the network capacity.

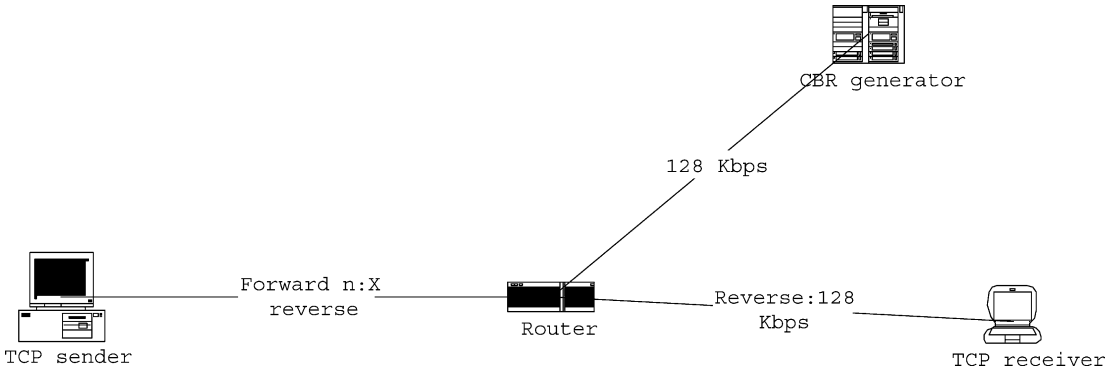

Fig. 5. Network topology for NS-2 simulation.

in the next section, we show that applying our scheme to TCP is more practical and simpler than the previous approaches described above.

7. Analysis and simulation

This section reports the results of two simulations. The first simulation is to calculate the means and the standard deviations of the forward and reverse delays in comparison with in TCP Reno. The goal of the first simulation is to show that our scheme tracks accurately in forward and reverse delay in a variety of networks. Taking advantage of simulation, we can get 'measured forward delay' by globally synchronized time. Using the protocol in Section 3, we can estimate the forward delay according to our derivation. By comparing these two values, we can validate that whether our scheme tracks forward delay accurately or not.

The second simulation is to evaluate the TCP performance with our estimation scheme. We apply the estimated one-way delay into TCP Reno and evaluate the performance with ns-2 simulator [15]. Through the second simulation, we demonstrate that using the forward delay, TCP achieves a significant improvement in the performance over the asymmetric networks.

7.1. Mean and standard deviation analysis

The network topology of our experiments for the asymmetric networks is shown in Fig. 5. The bandwidth of the reverse channel is 128 kbps, and the bandwidth of the forward channel is '$n \times 128$ kbps' (n is integer from 1 to 10), where n is the asymmetric ratio. The asymmetric ratio shows how much the bandwidth of the forward channel is larger than that of the reverse channel. For example, if an ADSL link consists of the forward channel of 8 Mbps and the reverse channel of 1 Mbps, its asymmetric ratio is 8. Also, if a link consists of the forward channel of 1 Mbps and the reverse channel of 5 Mbps, its asymmetric ratio is 0.2.

In our simulation study, the propagation delay of link is 50 ms. The segment size used is 1460 bytes, the router queue size is 50 packets, and the router queuing discipline is FIFO (First-In–First-Out). CBR (Constant Bit Rate) application is used for generating background traffic for the congestion in the reverse path.

Fig. 6 shows the result of the variation of estimated forward delay, measured forward delay, and RTT when the asymmetric ratio is 8 and there is 100 kbps' background traffic. Fig. 6(a) shows the comparison of the estimated forward delay value and TCP sender's RTT value.

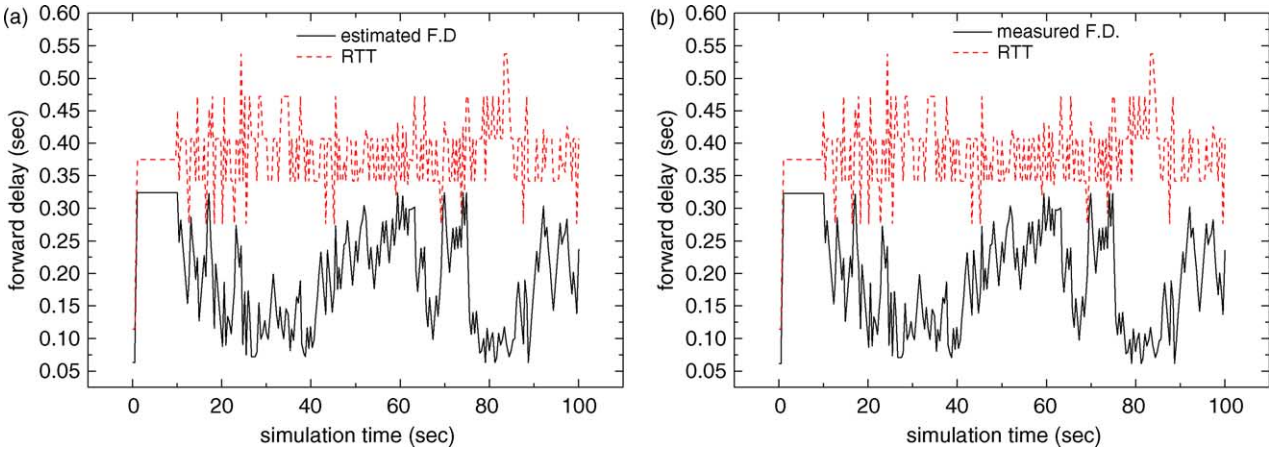

Fig. 6. Comparison of the forward delays with RTT estimated by TCP sender. (a) The estimated forward delay and RTT. (b) The measured forward delay and RTT.

Table 1
Comparison of estimated forward delay with measured forward delay without background traffic

Without background traffic				
	Estimad fd		Measured fd	
	Avg.	Std.	Avg.	Std.
0.1	7.617188	1.781096	7.564688	1.781096
0.3	7.075357	3.982878	7.07244	3.982878
0.5	4.74325	2.145377	4.742	2.145377
0.7	3.596594	1.36899	3.597129	1.368989
1	2.623649	0.838992	2.623649	0.838992
3	0.911326	0.167597	0.910493	0.167597
5	0.53564	0.074833	0.53464	0.074833
7	0.370909	0.042493	0.36989	0.042493
10	0.245947	0.021901	0.244822	0.021901

Table 2
Comparison of estimated forward delay with measured forward delay with 30 kbps background traffic

With background traffic(30 kbps)				
	Estimad fd		Measured fd	
	Avg.	Std.	Avg.	Std.
0.1	7.530813	1.79747	7.530778	1.79747
0.3	7.070144	3.981231	7.073061	3.981231
0.5	4.736642	2.142428	4.737892	2.142428
0.7	3.591718	1.366789	3.592253	1.366788
1	2.619188	0.837368	2.619188	0.837368
3	0.902363	0.166581	0.90153	0.166581
5	0.529321	0.075159	0.528321	0.075159
7	0.357442	0.044899	0.35637	0.044899
10	0.23584	0.028011	0.234715	0.028011

TCP sender continues to estimate RTT and smoothes them to be more conservative. Two graphs show RTT estimation of the sender is too conservative to adapt to the network state. And comparing Fig. 6(a) and (b), we can see that the changing shape of the delays[3] is almost same unlike RTT even though their absolute values have a very small difference. That is, we can confirm that our delay estimation scheme can track the bandwidth variation well even in the asymmetric networks.

Tables 1–3 summarize the mean value and the standard deviation of the measured and the estimated delays varying the asymmetric ratio from 0.1 to 10. Also, for the purpose of making the delay more variable, background CBR traffic is added to the reverse channel for 0–100 kbps. This configuration is helpful to show that our scheme keeps track of the one-way delay even though the network state dynamically changes, regardless of the degree of the asymmetry. Observation of the mean and the standard deviation says that the estimated delay and the measured delay are very close to each other. Standard deviation is almost same, and absolute values are different with at most 0.002 s. This says that our estimation scheme tracks the changing delay condition very accurately.

Small difference between the measured forward delay and the estimated forward delay come from the initial t_0 value. Even in the base case, this initial difference cannot be removed. However, as seen in the results, this difference is so small that our choice on the initial value is acceptable. Actually, the variation of the forward delay is affected much more on the network condition than the difference of t_0. This proves that our approach is well performed as expected from the derivation.

7.2. Simulation results of an application of our scheme

The same topology used in the mean analysis is also used in the simulation for TCP with our scheme. Almost all parameters are the same except the propagation delay of the reverse path, such as the segment size and the router queue size, etc. The reason to adjust the link delay of the reverse channel is to regulate the ACK arrival rate.

Fig. 7 shows the behavior in the slow start phase of our algorithm (TCP FI). For the purpose of focusing on the slow start phase, the simulation is performed for just 5 s, and the result is that TCP FI raises *cwnd* faster than Reno, regardless of the ACK arrival. Fig. 8 shows that TCP FI achieves better aggregate throughputs than TCP Reno by 17%. As the number of TCP connection increases, the aggregate throughput also increases consistently. In the point of view of the average performance, it is clear that TCP FI achieves the improvement in the throughput.

8. Discussion

Our delay estimation scheme tracks the changing network bandwidth very well. But its absolute value is affected by the initial fixed-point. As in simulation results, the standard deviation is almost same while the absolute value between the estimation value and the measured value has a little difference. We proposed a heuristic method to estimate the initial fixed-point. As a result, its error range is reduced and actually we could see that our heuristic

Table 3
Comparison of estimated forward delay with measured forward delay with 100 kbps background traffic

With background traffic (100 kbps)				
	Estimated fd		Measured fd	
	Avg.	Std.	Avg.	Std.
0.1	7.526613	1.799795	7.526255	1.799795
0.3	7.045031	3.963171	7.047947	3.963171
0.5	4.711177	2.12814	4.712427	2.12814
0.7	3.575713	1.359463	3.576248	1.359463
1	2.598705	0.829651	2.598705	0.829651
3	0.881034	0.163178	0.880201	0.163178
5	0.489119	0.076359	0.488119	0.076359
7	0.292937	0.071811	0.291866	0.07181
10	0.107206	0.054606	0.106081	0.054606

[3] Measured and estimated delays.

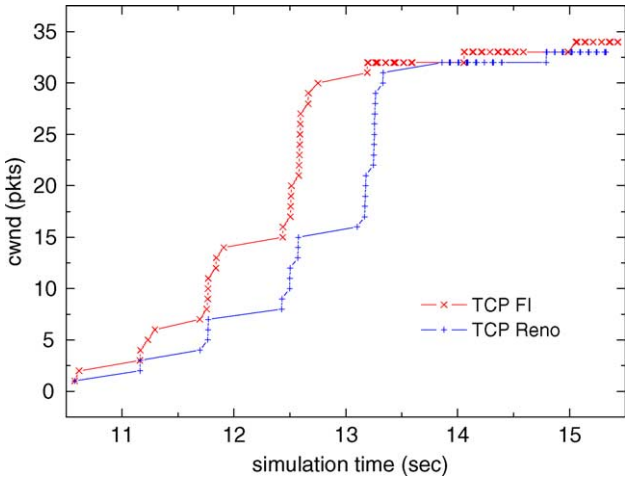

Fig. 7. Behaviors of TCP Reno and FI in the slow start phase.

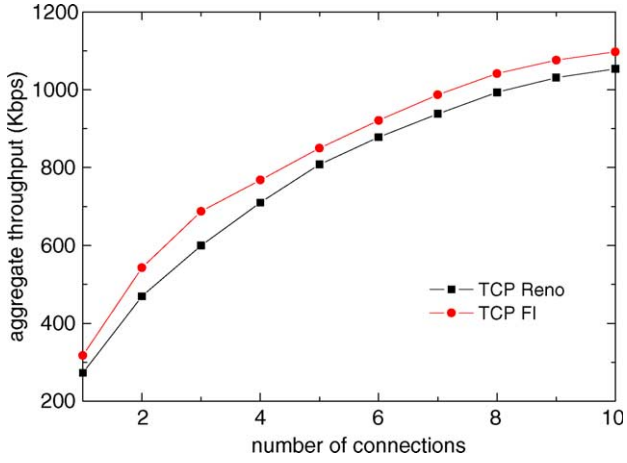

Fig. 8. Aggregate throughput of TCP Reno and FI.

approach provides reasonable estimation in a variety of network environments, as shown in Tables 1–3. However, the difference between the fixed point value of t_0 and real initial forward delay still exists. Also, this error range could be large if the propagation delays of two-way links are severely different or the network congestion is severe before making the session. So, to get more accurate result, we need more effective method that estimates or adjusts the initial fixed-point, t_0.

Timeout of TCP sender can be made based on the forward delay value. Current TCP implementations use the timeout value based on RTT. But we have to remind ourselves of the fact that RTT is ambiguous information for the network state. Therefore, the forward delay would be used as another criterion of RTO value.

9. Conclusion

This paper proposed a new scheme in which one-way delay can be analytically derived. A lot of work that tries to calibrate packet transit time focused on the clock synchronization and timestamp, but to our best knowledge any assured solution still does not exist. Therefore, we choose a different approach from previous work, which focus on tracking the delay accurately. As a result, we found that the difference of the sender-measured RTT and receiver-measured RTT draws the changing network condition and using this concept, designed a new delay estimation scheme.

Through simulations, we showed that our derivation of the one-way delay is very accurate and tracks the changes of the network state extremely well. By implementing the proposed scheme in TCP Reno, we verified the scheme's accuracy and using it, made improvement in TCP performance over the asymmetric networks. The result of applying the new scheme to TCP Reno in ns-2 is that the performance of TCP FI is better than regular TCP Reno. We believe that any version of TCP implementation can benefit from our one-way delay estimation scheme.

Acknowledgements

This work was supported by grant No. R01-2004-000-10588-0 from the Basic Research Program of the Korea Science and Engineering Foundation.

References

[1] V. Paxson, On calibrating measurements of packet transit times, in: Proceedings of SIGMETRICS 1998, June 1998.
[2] S. Moon, P. Skelly, D. Towsley, Estimation and removal of clock skew from network delay measurements, in: Proceedings of INFOCOM 1999, March 1999.
[3] K. Anagnostakis, M. Greenwald, R. Ryger, cing: Measuring network-internal delays using only existing infrastructure, in: Proceedings of INFOCOM 2003, April 2003.
[4] D. Mills, Improved algorithms for synchronizing computer network clocks, IEEE/ACM Transactions on Networking 3 (3) (1995).
[5] D. Mills, Network Time Protocol (Version 3): Specification, Implementation and Analysis, RFC 1305, Network Information Center, SRI International, Menlo Park, CA, 1992.
[6] D. Mills, Modelling, Analysis of Computer Network Clocks, Technical Report 92-5-2, Electrical Engineering Department, University of Delaware, May 1992.
[7] J. Postel, Transmission Control Protocol, RFC 793, September 1981.
[8] V. Raisinhani, A. Patil, S. Iyer, Mild aggression: a new approach for improving TCP performance in asymmetric networks, in: Proceedings of AMOC 2000, October 2000.
[9] I. Ming-Chit, D. Jinsong, W. Wang, Improving TCP performance over asymmetric networks, ACM Computer Communication Review 30 (3) (2000).
[10] M. Allman, S. Dawkins, D. Glover, J. Griner, D. Tran, T. Henderson, J. Heidemann, J. Touch, H. Kruse, S. Ostermann, K. Scott, J. Semke, Ongoing TCP Research Related to Satellites, RFC 2760, February 2000.

[11] T. Henderson, R. Katz, Transport protocols for Internet-compatible satellite networks, IEEE Journal on Selected Areas in Communication 17 (2) (1999).
[12] H. Balakrishnan, V. Padmanabhan, R. Katz, The effects of asymmetry on TCP performance, in: Proceedings of Mobicom 1997, September 1997.
[13] M. Allman, V. Paxson, On estimating end-to-end network path properties, in: Proceedings of SIGCOMM 1999, September 1999.
[14] V. Paxson, End-to-end routing behavior in the Internet, in: Proceedings of SIGCOMM 1996, August 1996.
[15] ns2 Network Simulator version 2.26, http://www.isi.edu/nsnam/ns, 2003.