Analytic End-to-End Estimation for the One-Way Delay and Its Variation

Jin-Hee Choi and Chuck Yoo
Department of Computer Science and Engineering
Korea University
Email: {jhchoi, hxy}@os.korea.ac.kr
Telephone: +82-2-3290-3639
Fax: +82-2-922-6341

***Abstract*— Delay estimation is a difficult problem in computer networks. 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. Accurate one-way delay estimation becomes crucial because it serves a very important role in network and application design.**

This paper proposes a new scheme to estimate one-way delay and its variation. The scheme calibrates estimated one-way delay so in a brief duration as to be used in many protocols that adjust their behavior depending on the network condition. We analytically derive one-way delay, forward and reverse delay respectively, and show that our one-way delay estimation is much more accurate than RTT estimation by simulation.

I. 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 it is guaranteed that there is a global time synchronization between the sender and the receiver. If it 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. With a considerable amount of research efforts have been devoted[1][2][3], the clock synchronization problem is not solved to the level suitable for calibrating packet's forward and reverse delay[4]. 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[5]. However, in today's networks such as cable modem networks, direct broadcast satellite networks, and ADSL networks, the bandwidth in the forward direction is often larger by orders of magnitude than that of the reverse link[6][7][8][9]. Furthermore, a large-scale study of Internet routing has found that paths through the Internet are often asymmetric[10][11], 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 the use of RTT difficult in place of one-way delay.

This paper proposes an accurate delay estimation scheme that analytically calculates the forward and reverse delay respectively. Our scheme gets estimated delay in a short time. Therefore, it can be used in many protocols that adjust their behavior depending on the network condition.

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 estimation scheme. Section 3 describes the new delay estimation scheme. Section 4 explains the implementation protocol for using our scheme and the implementation details. In Section 5, we show the simulation results and analyze the reason. We discuss the results and an unsolved problem of our method in Section 6. Finally, Section 7 concludes the paper.

II. Motivation and background

Motivation: Before we introduce our estimation scheme, let us first examine a sample that shows the difference of real forward delay and round trip time at the TCP sender. This sample is a result of the simulation in Section 6, and it is illustrated in Figure 1.

In this graph, X-axis is simulation time, and the Y-axis is the one-way delay and the $\frac{RTT}{2}$. The forward delay is calculated by subtracting the sender's timestamp in each packet header from the receiver's clock, under assumption of clock synchronization. Figure1's observation says that there can be the large difference of the forward delay and the $\frac{RTT}{2}$ even in symmetric network, and such wrong assumption (RTT is good approximation of the forward delay) can misuse it, and consequently make harm in the protocol performance even though RTT is very important information about network condition in the end system. It is natural that one-way delay is proposed for the solution of this problem. The attempts in previous works to obtain 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

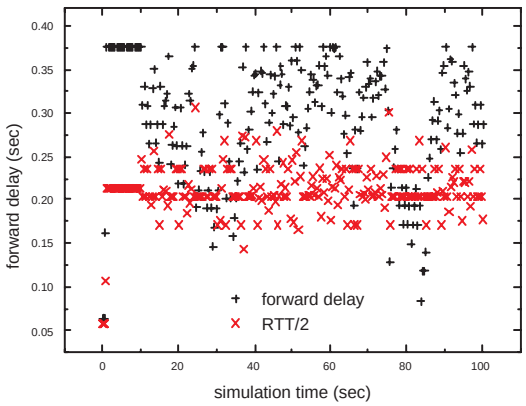

Fig. 1. Forward delay and $\frac{RTT}{2}$.

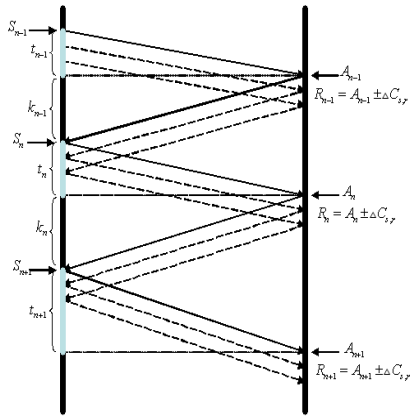

Fig. 2. Typical packet exchange of TCP sender and receiver.

clocks. Many schemes to use physical clock synchronization algorithm is included in it, but this scheme is difficult to be applied to massive distributed computing environment. The other is to get a re-processing value by removing the clock skew from the one-way delay calculated by the difference of two timestamps. Paxson[1] and Moon[2] addressed this problem and proposed a solution respectively. However, these schemes cannot be practically applied to the transport protocol that has to adjust its behavior depending on the change of the network condition because they require the re-processing time to obtain the one-way delay from measurement.

Our proposed 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 gets estimated delay in a short time; it can be used in many protocols that adjust their behavior depending on the network condition.
- It is so simple as to be implemented practically.

Thus, we believe it would enhance the performance of many works in fields of QoS(Quality-of-Service), real time system and network measurement, etc.

Basic clock terminology: Let us define basic terminology for discussing the characteristics of the clocks used in following study. Mill[4] 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.

III. Delay Derivation

Our scheme assumes that the receiver responds with ACK immediately for each arriving packet to the sender. (i.e. TCP receiver does not employ delayed ACK.) Figure 2 shows a typical interchange of packets and ACKs.

Let us introduce the terminology for clocks, timestamp and delays used in our estimation scheme.

- C_s: sender clock.
- C_r: receiver clock.
- S_n: the transmission time of the sender for n-th packet according to C_s.
- R_n: the arrival time of n-th packet at the receiver according to C_r.
- A_n: the arrival time of n-th packet at the receiver according to C_s.
- t_n: the forward delay of the n-th packet according to C_s; $A_n - S_n$.
- k_n: the reverse delay of the n-th 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:
  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).

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

- RTT measured by receiver:
  Receiver measures RTT by the difference of the time of sending the previous ACK and the current ACK. So, it denotes $RTT(r, n) = R_n - R_{n-1}$, 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).

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

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

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

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

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

$$\sum_{i=1}^{n} RTT(s, i+1) - \sum_{i=1}^{n} RTT(r, i) = k_n - k_0 \tag{4}$$

When n is 0, equation (1), $RTT(s, n+1) = t_n + k_n$ becomes $RTT(s, 1) = t_0 + k_0$

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

$$\begin{aligned} k_n &= RTT(s, n+1) - t_n \\ k_0 &= RTT(s, 1) - t_0 \end{aligned}$$

Thus, we get

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

Finally, we have forward delay t_n,

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

Also, we have reverse delay, k_n

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

Eq.(6) and eq.(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.

IV. Protocol Implementation

To implement eq.(6), we need to calculate three values: initial fixed point value, sender-measured RTT, and receiver-measured RTT. The pseudo-code of our protocol is described in Figure 3.

- t_0 : Although eq.(6) shows that the one-way delay can be analytically derived, the accuracy of the delay estimation is affected by t_0. So, a question is how to get proper t_0. When TCP session is created, if there is nearly no congestion in networks, using $\frac{RTT}{2}$ as t_0 shows acceptable error range compared with absolute value of t_0. However, it is difficult to predict the network condition when the TCP session is created. Thus naturally, such a scheme does not reflect the network congestion that occurs before the TCP session. So we need a scheme to reduce the error range up to the level of acceptance in end-to-end protocol because it is impossible to get the accurate t_0 without any information in initial phase.
  We can define a variable d that is defined as the difference of forward and reverse delay ($t_0 - k_0 = d$).
  - $d > 0$
    We get $2t_0 > RTT(s, 0) > 2k_0$ by $t_0 > k_0$. By summing t_1 to $RTT(s, 0) > 2k_0$ on both sides, we also get $t_1 + RTT(s, 0) > t_1 + 2k_0$. Now, we can rewrite new range of t_0 as follows by using expressions, $t_1 + k_0 = RTT(r, 0)$ and $t_1 = t_0 + RTT(s, 1) - RTT(r, 1)$.

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

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

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

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 t_0;
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. $t_n = t_0 - fd\_diff$;
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

– $d = 0$

Because t_0 and k_0 are same,

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

We can limit the field of the difference of estimated and absolute values with the above expressions. To our knowledge, however, there is no practical method to know the sign of d. Therefore, we choose a heuristic scheme to do it.

This heuristic method uses the property of one-way delay in Internet, which is that queuing delay tends to change steady and is dominant part in 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 $\frac{RTT(s,1)-RTT(r,1)+RTT(r,0)+2RTT(s,0)}{4}$ as t_0 (the range average in case of $d > 0$). The reverse, if the ratio is smaller than β, we choose $\frac{RTT(s,1)-RTT(r,1)+RTT(r,0)}{4}$ as t_0 (the range average in case of $d < 0$). Also, if the ratio is larger than β and smaller than α, we can just choose $\frac{RTT(s,0)}{2}$ as t_0. α and β are approximation to middle values in 0, $\frac{RTT(s,0)}{2}$ and $RTT(s,0)$. 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.

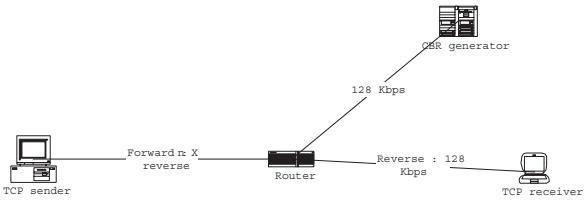

Fig. 4. Network Topology for NS-2 simulation

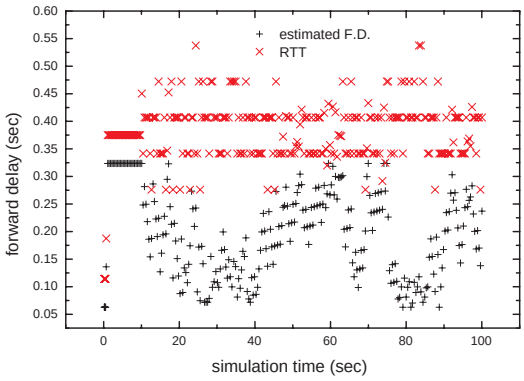

Fig. 5. Comparison of estimated forward delay with RTT estimated by TCP sender

- $RTT(s,I)$: All existing versions of TCP periodically measure RTT to calculate RTO or sending rate. Therefore, it is easy for sender to measure RTT.

- $RTT(r,I)$: The most intuitive way of measuring RTT at receiver is to measure interval of ACKs. Receiver has to write such RTT values into the header of ACK packet since the sender needs the difference of 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.

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

V. Analysis and Simulation

This section reports the results of simulation. The simulation is to calculate the means and standard deviations of the forward and reverse delays in comparison with TCP Reno. The goal of the first simulation is to show that the proposing protocol tracks accurately in forward and reverse delay in

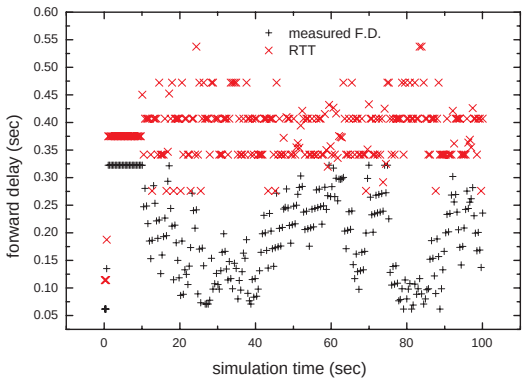

Fig. 6. Comparison of measured forward delay with RTT estimated by TCP sender

Without backgroud traffic				
	estimated 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 I

COMPARISON OF ESTIMATED FORWARD DELAY WITH MEASURED FORWARD DELAY WITHOUT BACKGROUND TRAFFIC

With background traffic(30Kbps)				
	estimated 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

TABLE II

COMPARISON OF ESTIMATED FORWARD DELAY WITH MEASURED FORWARD DELAY WITH 30KBPS BACKGROUND TRAFFIC

various networks. Taking advantage of simulation, we can get measured forward delay by measuring globally synchronized time. Using the protocol in section 4, we can estimate the forward delay follows our derivation. By comparing these two value, we can validate that whether our proposing scheme tracks forward delay accurately or not. Additionally, another comparison is made by comparing the existing TCP's RTT estimation value and measured forward delay value.

A. Mean and standard deviation analysis

The network topology of our experiments for asymmetric networks is shown in Figure 4. The bandwidth of the reverse channel is 128Kbps, and the bandwidth of the forward channel is (n is integer from 1 to 10), where n is asymmetric ratio. Asymmetric ratio is a value that shows how much the bandwidth of forward channel is larger than that of reverse channel. For example, if an ADSL link consists of forward channel of 8Mbps and reverse channel of 1Mbps, its asymmetric ratio is 8. Also, if a link consists of forward channel of 1Mbps and reverse channel of 5Mbps, its asymmetric ratio is 0.2. By using simulation experiments, we can get synchronized global time value between sender and receiver.

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

Figure 5 and 6 show the result of the variation of estimated forward delay, measured forward delay, and RTT when the asymmetric ratio is 8 and there is 100Kbps's background traffic. Figure 5 shows the comparison of estimated forward delay value and TCP sender's RTT value. TCP sender estimates RTT and smoothes them to be more conservative. Two graph shows RTT estimation of TCP sender is too conservative to adapt to the network status.

Figure 6 shows the comparison of measured forward delay value and TCP sender's RTT value. Comparing two figures, the changing shape of the delays is almost same 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 asymmetric networks.

Table 1, 2, 3 summarize the mean value and the standard deviation varying the asymmetric ratio from 0.1 to 10. Also, it shows the mean and the standard deviation of estimated forward delay-i.e. from derivation-and measured forward delay-i.e. by time synchronization is assumed-when a background CBR traffic is added from 0 to 100 Kbps. The purpose is to demonstrate the influence of the background traffic of reverse channel on estimation results. Observation of the mean and the standard deviation says that the estimated forward delay and the measured forward delay are very close to each other. Standard deviation is almost same value although absolute values are different from 0.002 to 0.1. This says that our estimation scheme tracks the changing delay condition very accurately.

Small differences between measured forward delays and estimated forward delay comes from the initial t_0 value. Even in the best case, this initial difference cannot be disappeared. However, as seen in results, this difference is so small that our expectation on the initial value, $RTT/2$ is proper value. Actually, The variation of forward delay is affected much more on network condition than the difference of t_0. This says that our approach is well performed as expected.

With background traffic(100Kbps)				
	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

TABLE III

COMPARISON OF ESTIMATED FORWARD DELAY WITH MEASURED FORWARD DELAY WITH 100KBPS BACKGROUND TRAFFIC

VI. DISCUSSION

Our delay estimation scheme tracks quiet well the changing network condition, but its absolute value is affected by the base forward delay. As in simulation results, standard deviation is almost same, but absolute value between estimated value and measured forward delay value has a little difference. We proposed a method to estimate the base forward delay. As a result, its error range is reduced and actually we could see that our heuristic approach provides reasonable estimation in Internet environments. 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 novel method that estimates or adjusts the base forward delay, t_0.

Many implementations of TCP use delayed ACK scheme. With delayed ACK scheme, TCP has to wait for expiration of delayed ACK timer even when it receives the TCP segment (generally, the timeout value is fixed to 500 ms). Since our estimation scheme assumes that, the receiver responds to the segment by sending ACK packet as soon as a segment arrives, the scheme could be corrupted by this delayed ACK. The simple way to solve it is that the receiver writes the value of the timeout into the header of ACK. However, although delayed ACK scheme is used, ACK will be generated at least of every other data packets. In window-based transmission protocol, like TCP, sender transmits multiple segments at the same time, delayed ACK does not affect as much as consideration.

Timeout of TCP sender can be made based on the forward delay value. Current TCP implementations use RTT based timeout values. Therefore, forward delay can be used as a another criterion of retransmission timeout value.

Proposing scheme can be applied to any kinds of protocol uses RTT estimation. If both sender and receiver can measure the RTT, our scheme can be applied in various kind of network environments. Additionally, RTT value is a kind of open questions in computer networks, and it can be applied to dynamically in various future protocols.

VII. CONCLUSION

This paper proposes a new scheme in which one-way delay can be analytically derived. Lots of works that try to calibrate packet transit times bring focus into clock synchronization and timestamp but in our knowledge, still assured solution does not exist. Therefore, we choose a different approach from previous works, 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 show that our derivation of one-way delay is very accurate and tracks the changes of the network condition extremely well. Implementing proposed scheme to TCP Reno , we verified the scheme's accuracy. We believe that any version of TCP implementation using RTT-based adaptation can benefit from our new scheme of one-way delay estimation and it can help many existing problems to be solved in computer networks.

ACKNOWLEDGEMENT

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

REFERENCES

[1] V. Paxson: *On Calibrating Measurements of Packet Transit Times*, In Proc. SIGMETRICS 1998, June 1998.
[2] S. Moon, P. Skelly, and D. Towsley: *Estimation and Removal of Clock Skew from Network Delay Measurements*, In Proc. INFOCOM 1999, March 1999.
[3] K. Anagnostakis, M. Greenwald, and R. Ryger: *cing: Measuring Network-Internal Delays using only Existing Infrastructure*, In Proc. INFOCOM 2003, April 2003.
[4] D. Mills: *Modelling and Analysis of Computer Network Clocks*, Technical Report 92-5-2, Electrical Engineering Department, University of Delaware, May 1992.
[5] J. Postel: *Transmission Control Protocol*,RFC 793, Sept. 1981.
[6] V. Raisinhani, A. Patil, and S. Iyer: *Mild Aggression: A new approach for improving TCP Performance in Asymmetric Networks*, In Proc. AMOC 2000, Oct. 2000.
[7] M. Allman, S. Dawkins, D. Glover, J. Griner, D. Tran, T. Henderson, J. Heidemann, J. Touch, H. Kruse, S. Ostermann, K. Scott, and J. Semke: *Ongoing TCP Research Related to Satellites*, RFC 2760, Feb. 2000.
[8] T. Henderson and R. Katz: *Transport Protocols for Internet-Compatible Satellite Networks*, IEEE Journal on Selected Areas in Communication, Vol 17, No. 2, Feb. 1999.
[9] H. Balakrishnan, V. Padmanabhan, and R. Katz: *The Effects of Asymmetry on TCP Performance*, In Proc. Mobicom 1997, Sep. 1997.
[10] M. Allman and V. Paxson: *On Estimating End-to-End Network Path Properties*, In Proc. SIGCOMM 1999, Sep. 1999.
[11] V. Paxson: *End-to-End Routing Behavior in the Internet*, In Proc. SIGCOMM 1996, Aug. 1996.
[12] *ns2 Network Simulator version 2.26*, http://www.isi.edu/nsnam/ns, 2003.