Inferring TCP Congestion Control Algorithms by Correlating Congestion Window Sizes and their Differences

Toshihiko Kato, Atsushi Oda, Shun Ayukawa, Celimuge Wu, Satoshi Ohzahata
Graduate School of Information Systems
University of Electro-Communications
Chofu-shi, Tokyo, Japan
e-mail: kato@is.uec.ac.jp, oda@net.is.uec.ac.jp, s.aykw@net is.uec.ac.jp, clmg@is.uec.ac.jp, ohzahata@is.uec.ac.jp

Abstract— **Recently, according to the diversification of network environments, a lot of TCP congestion control mechanisms have been introduced. They define how a TCP sender increases the congestion window (*cwnd*) during no congestion and decreases *cwnd* when it detects congestion. Since the congestion control algorithms affect the performance of the Internet, it is important to know which algorithms are used widely. This paper proposes a scheme to infer the algorithm for individual TCP flows by examining the packet trace passively captured for the flows. Our scheme estimates *cwnd* values in RTT intervals and correlates the *cwnd* values and the differences of consecutive *cwnd* values. Our scheme aims to infer many of recently proposed congestion control algorithms, which have been out of scope in the conventional passive approaches. Our scheme adopts a simple approach just correlating *cwnds* and their differences, in contrast with the conventional approaches which estimate the internal TCP behaviors based on packet traces. This paper describes the details of our scheme and shows the results where our scheme is applied to an iPhone TCP communication.**

Keywords- TCP congestion control algorithms; passive monitoring; congestion window .

I. INTRODUCTION

Since the congestion control mechanism came to be used in Transmission Control Protocol (TCP) [1], only a few algorithms, such as Tahoe, Reno and NewReno [2], were used commonly for a long time. In the congestion control, a TCP sender transmits data segments under the limitation of the congestion window (*cwnd*) maintained within the sender, beside the advertised window reported from a TCP receiver. The value of *cwnd* increases as a sender receives ACK segments and is decreased when it detects congestions. How to increase and decrease *cwnd* is the key of congestion control algorithm. The early-stage algorithms mentioned above are summarized as an additive increase and multiplicative decrease (AIMD) because *cwnd* increases linearly and is reduced in an exponential fashion.

According to the diversification of network environments, many TCP congestion control algorithms have emerged [3]. For example, High Speed (HS) TCP [4], and CUBIC TCP [5] are designed for high speed and long delay networks. On the other hand, TCP Westwood [6] and its descendants are designed for lossy wireless links. While the algorithms mentioned so far are based on the packet losses, TCP Vegas [7] and FAST TCP [8] trigger congestion control against an increase of round-trip time (RTT). TCP Veno [9] and TCP Illinois [10] combine loss based and delay based approaches such that congestion control is triggered by packet losses but the delay determines how to increase *cwnd*.

The TCP congestion control algorithms affect the performance of the Internet, and so it is important to know which algorithms are used widely. Since the congestion control algorithm is implemented within a TCP sender, it cannot be identified from observable parameters in TCP segments. Instead, a tester which infers the algorithm needs to estimates internal behaviors of TCP senders from their input/output interactions.

The approaches to infer the congestion control algorithm are categorized into two groups. One is the passive approach where passively collected packet traces are examined to measure TCP behaviors. This approach has some limitations in the testing ability, but is non-intrusive and requires no additional equipment for measurement. The other is the active approach in which an active tester sends test inputs to a target node and checks the replies. This approach can perform a more comprehensive test than the passive one, but is limited to the case where a tester communicates with a node to be tested.

So far, several studies are proposed for both approaches [11]-[16]. However, as for the passive approach, there are no proposals on inferring the recently introduced algorithms. In this paper, we propose a new scheme based on the passive approach. The proposed scheme aims to infer many of recent congestion control algorithms and adopts a simpler methodology than the conventional studies.

The rest of this paper consists of the following sections. Section 2 surveys the related works specifically. Section 3 proposes our scheme. Section 4 gives some examples where our scheme is applied to an iPhone TCP communication. Section 5 gives the conclusions of this paper.

II. RELATED WORKS

As for the passive approach, TCPanaly [11] is one of the early stage research activities. It analyzes packet traces and tries to decide which implementation of TCP best matches the connection being observed.

Jaiswel et al. [12] adopted a similar approach with TCPanaly and proposes the TCP flavor identification among Tahoe, Reno and NewReno. Its basic idea is to construct three kinds of “replicas” of the TCP sender’s state machine for individual TCP connections observed at the measurement point. These replicas are for Tahoe, Reno and NewReno. For a segment sent by a TCP sender, each replica checks whether the segment is allowed or not. The numbers of violations are maintained and the TCP flavor with minimum number of violations is selected for the connection.

Oshio et al. [13] estimates the changes of *cwnd* values and extracts features, such as ratio of *cwnd* increase being one and so on. Based on these features, it discriminates one of two different versions randomly selected from thirteen TCP versions implemented in the Linux operating system.

Qian et al. [14], on the other hand, focuses on the extraction of statistical features based on the monitoring of one direction of TCP communications. They focused on the size of initial congestion window, the relationship between the retransmission rate and the time required to transfer a fixed size of data, which is used for detecting the irregular retransmissions, and the extraction of flow clock to find the TCP data transmission controlled by the application or link layer factors.

As an example of the active approach, TBIT [15] was developed to characterize the TCP behavior of major web servers. It checks the initial window size by not acknowledging any data segments sent by the server at the first data transfer. It also detects the congestion control algorithm by dropping two data segments (not acknowledging them) within one window. This discriminates Tahoe, Reno and NewReno.

CAAI [16] proposes the scheme to actively identify the TCP algorithm of a remote web server. It can identify all default TCP algorithms, such as AIMD and CUBIC, and most non-default TCP algorithms of major operating system families. It makes a web server send 512 data segments under the controlled network environment with specific RTT and observes the number of data segments contiguously transmitted without receiving any ACK segments. It then estimates the window growth function and the decrease coefficient, and using those estimations, determines the TCP algorithm for an individual web server.

As described above, CAAI proposes the inference of TCP congestion control algorithms used widely today, but no studies from the standpoint of passive approach. This paper proposes a passive monitoring based approach for inferring many of the TCP versions available today.

III. PROPOSAL OF OUR SCHEME

A. *Design principles*

The TCP congestion control algorithms have two parts. One is a part where a TCP sender increases *cwnd* at receiving an ACK segment acknowledging new data segments. The other is a part where a TCP sender decreases *cwnd* when it detects network congestion through retransmitting any data segments or perceiving an increase of RTT.

Our scheme to infer the congestion control algorithm is designed based on the following principles.

- Our scheme focuses on the increasing part of *cwnd*.
- It uses changes of the values of *cwnd* at individual RTT intervals.

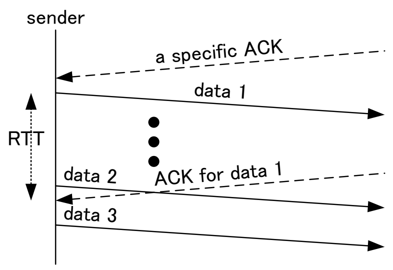


Figure 1. Principle for *cwnd* estimation.

- It estimates the value of *cwnd* at a moment when a TCP sender receives a specific ACK segment as the total size of inflight data segments, which are sent but not acknowledged, just before the TCP sender receives the ACK segment one RTT later than the first ACK. Fig. 1 shows this mechanism. Fig. 1 supposes that a sender receives *a specific ACK* and then sends *data 1*. After one RTT, the sender receives *ACK for data 1*. *Data 2* is the data segment sent out just before the sender receiving *ACK for data 1* and *data 3* is the data segment sent out just after the acknowledgment. Here, our scheme estimates the value of *cwnd* when the sender receives *a specific ACK* as
$$seqence\ number\ of\ data\ 2 + its\ length - ACK\ number\ of\ a\ specific\ ACK,$$ that is, $$seqence\ number\ of\ data\ 3 - ACK\ number\ of\ a\ specific\ ACK.$$
- The packet trace used in the inference may be captured in the middle of network. Therefore, in general, the packet sequence in the trace is different from the sequence in which the relevant TCP sender sends and receives packets. Our scheme needs to estimate the packet sequence in the TCP sender from that in the packet trace. For this purpose, our scheme utilizes the TCP time stamp option in TCP segments.
- Our scheme estimates a sequence of *cwnd* values observed in every RTT interval. We denote this sequence as $\{cwnd_i\}$. Then, a sequence of differences of consecutive *cwnd* values, $\{\Delta cwnd_i\}$, is defined by (1).
$$\Delta cwnd_i = cwnd_{i+1} - cwnd_i \qquad (1)$$
- In the end, our scheme evaluates the correlation of the two sequences, $\{cwnd_i\}$ and $\{\Delta cwnd_i\}$, by plotting them. The graphs depend on the congestion control algorithms.

It should be noted that, since our scheme does not require any tracking of TCP internal status, it is possible to infer the congestion control algorithms more easily than the conventional proposals.

The reason we adopt *cwnd* values at RTT intervals is as follows. First of all, many congestion control algorithms, such as Vegas and HS TCP, define a procedure for increasing *cwnd* at a RTT interval. Some algorithms, such as AIMD, specify a procedure for receiving individual ACK segments, but the purpose of those algorithms is the change of *cwnd* in a RTT interval. So, focusing the *cwnd* values at

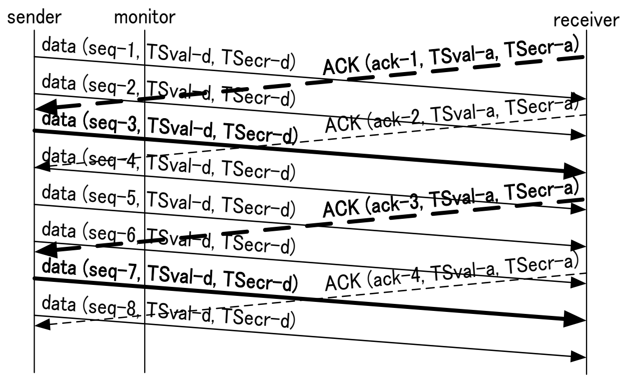


Figure 2. Estimation of *cwnd* associated with one RTT.

RTT intervals is considered as appropriate for reflecting the purpose of congestion control algorithms.

The next point is that the algorithms define *cwnd* values in a byte but data segments are sent in the unit of maximum segment size (MSS). Therefore, the passive approach can detect the change of *cwnd* values in the order of MSS. On the other hand, many algorithms change *cwnd* values in the order of MSS during a RTT interval.

B. Methodology for estimating cwnd at RTT intervals

As mentioned above, the time associated with individual captured segments in a packet trace is not the exact time when the data sender sent or received those segments. So, our scheme estimates a *cwnd* associated with one RTT interval in the following way.

- First, our scheme focuses on an ACK segment in the packet trace, for example, *ACK with ack-1* in Fig. 2.
- Next, it looks for the first data segment whose TSecr (Time Stamp Echo Reply) is equal to TSval (Time Stamp Value) of the ACK segment we are focusing on, *data with seq-3* in Fig. 2.
- Our scheme then looks for the first ACK segment acknowledging this data segment, in this case, *ACK with ack-3*.
- As the fourth step, it looks for the data segment whose TSecr is equal to TSval of the second ACK segment. In the example of Fig. 2, this corresponds to *data with seq-7*.

After these steps, our scheme estimates that the first data segment, *data with seq-3*, and the second ACK segment, *ACK with ack-3*, construct a RTT relationship. Based on this estimation, our scheme estimates that *cwnd* in the unit of MSS at the moment of receiving *ACK with ack-1* is equal to (2).

$$\frac{seq\text{-}7 - ack\text{-}1}{MSS} \qquad (2)$$

C. Applying our scheme to AIMD

In the AIMD congestion control algorithm, *cwnd* is increased each time the TCP sender receives an ACK segment acknowledging new data. The increase is one segment during the slow start phase, and $\frac{1}{cwnd}$ segments during the congestion avoidance phase. During one RTT, *cwnd* of data segments are sent and acknowledged. Therefore, in the slow start phase, *cwnd* is increased by the value of *cwnd* (*cwnd* is doubled). This means that

$$\Delta cwnd_i = cwnd_i.$$

On the other hand, in the congestion avoidance phase, *cwnd* is increased by one segment during one RTT. So, in this phase,

$$\Delta cwnd_i = 1.$$

So, plotting *cwnd* and $\Delta cwnd$ generates the graph in Fig. 3. In this figure, it is assumed that the initial congestion window is one MSS, and that the slow start continues until *cwnd* is 16 followed by the congestion avoidance.

D. Applying our scheme to TCP Vegas

TCP Vegas detects congestion by the increase of RTT. It measures the minimal RTT during the connection lifetime. With the current values of *cwnd* and RTT, it estimates the buffer size in the bottleneck node as (3).

$$BufferSize = cwnd \times \frac{RTT - RTT_{min}}{RTT} \qquad (3)$$

Vegas uses this *BufferSize* for the control in the congestion avoidance phase in the following way.

- If $BufferSize < \alpha$, then *cwnd* is increased by one MSS. (In the Linux implementation, α is less than 2.)
- If $BufferSize > \beta$, then *cwnd* is decreased by one MSS. (In the Linux implementation, β is more than 4.)
- If $\alpha \leq BufferSize \leq \beta$, then the system is considered to be in a steady state and no modification to *cwnd* is applied.

This examination is done at every RTT interval. Therefore, the difference of *cwnd* at RTT interval, $\{\Delta cwnd_i\}$, is

$$\Delta cwnd_i = 1, 0 \; or -1$$

in the congestion avoidance phase.

As for the slow start phase, *cwnd* is increased every other RTT. This means that $\{\Delta cwnd_i\}$ is

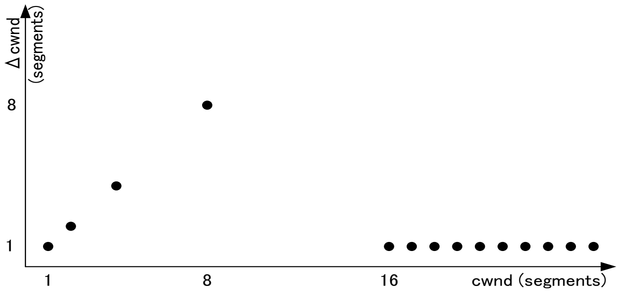


Figure 3. Applying to AIMD.

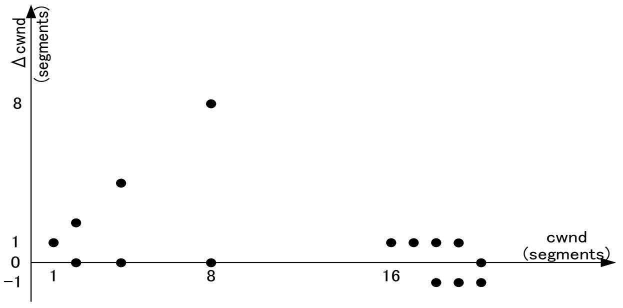


Figure 4. Applying to TCP Vegas.

$$\Delta cwnd_i = cwnd_i \ or \ 0$$

in this phase. So, plotting $cwnd$ and $\Delta cwnd$ generates the graph in Fig. 4 for TCP Vegas. In this figure, it is assumed that the initial congestion window is one MSS, that the slow start continues until *cwnd* is 16, and that *BufferSize* increases when *cwnd* is 20.

E. *Applying our scheme to TCP Veno*

The Veno (VEgas and ReNO) algorithm uses the Vegas estimate in order to limit the increase of *cwnd* during the congestion avoidance phase. If the Vegas buffer estimate shows excessive buffer utilization (i.e., $BufferSize > \beta$), a TCP sender increases *cwnd* by one for every two RTT.

This means that the increase of *cwnd* during the congestion avoidance phase is

$\Delta cwnd_i = 1$ during no congestion, and

$\Delta cwnd_i = 1 \ or \ 0$ during congestion.

As a result, the plotting of *cwnd* and $\Delta cwnd$ will be as the graph in Fig. 5 for Veno. In this figure, it is assumed that the initial congestion window is one MSS, that the slow start continues until *cwnd* is 16, and that congestion occurs when *cwnd* is 20.

F. *Applying our scheme to HS TCP*

The HS TCP changes the increase coefficient α according to the current size of cwnd. Here, α defines how many segments are added to *cwnd* for one RTT in the congestion avoidance phase. When cwnd is less than or equal to 38 segments, α is 1, which has the same behavior as the traditional AIMD. If *cwnd* is more than 84K segments, α is 70. Between 38 and 84K segments, α is interpolated from 1 and 70 linearly.

In the slow start phase, HS TCP adopts the limited slow start, which bounds the maximum increase step during this phase to 100 segments.

These specifications give the plotting of *cwnd* and $\Delta cwnd$ as shown in Fig. 6 in the form of semilog graph. In the graph, the congestion avoidance is started from *cwnd* of 32, and the relationship between the consecutive *cwnds* is defined as in (4) for $cwnd_i$ which is between 38 and 8700.

$$cwnd_{i+1} = cwnd_i + \frac{70-1}{8700-38}(cwnd_i - 38) + 1 \quad (4)$$

G. *Applying our scheme to CUBIC TCP*

CUBIC TCP defines cwnd as a cubic function of elapsed time T since the last congestion event. Specifically, it defines cwnd by (5).

$$cwnd = C\left(T - \sqrt[3]{\beta \cdot \frac{cwnd_{max}}{C}}\right)^3 + cwnd_{max} \qquad (5)$$

Here, C is a predefined constant, β is a coefficient of multiplicative decrease in the congestion control, and $cwnd_{max}$is the value of *cwnd* just before the loss detection in the last congestion event.

From this equation, the increase of *cwnd* during one RTT can be obtained approximately by (6).

$$RTT \cdot \frac{d(cwnd)}{dT} = RTT \cdot 3C\left(T - \sqrt[3]{\beta \cdot \frac{cwnd_{max}}{C}}\right)^2 \qquad (6)$$

By using (5), (6) is represented as a function of *cwnd* as in (7).

$$RTT \cdot \frac{d(cwnd)}{dT} = 3RTT \cdot \sqrt[3]{C}\left(\sqrt[3]{cwnd - cwnd_{max}}\right)^2 \quad (7)$$

This result gives the plotting of *cwnd* and $\Delta cwnd$ as in Fig. 7. Here, it is assumed that $cwnd_{max}$ is 0 in the slow start phase and $3RTT \cdot \sqrt[3]{C} = 1$.

H. *Applying our scheme to TCP Illinois*

TCP Illinois changes the increase coefficient of *cwnd*, α, according to the queuing delay. The queuing delay is measured as the increase of RTT from the minimum RTT for the connection. Depending on the queuing delay, α changes from 0.1 segments to 10 segments. The value of α is updated once per every RTT. Therefore, the plotting of *cwnd* and $\Delta cwnd$ will be given as in Fig. 8. In this figure, it is assumed that the initial congestion window is one MSS,

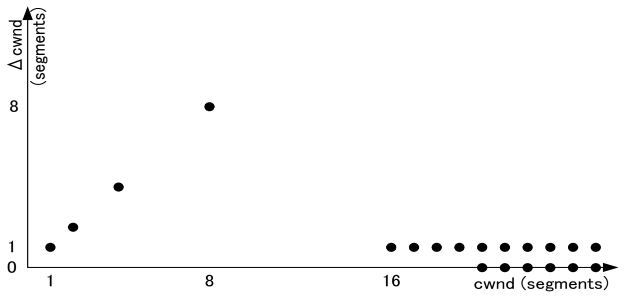


Figure 5. Applying to TCP Veno.

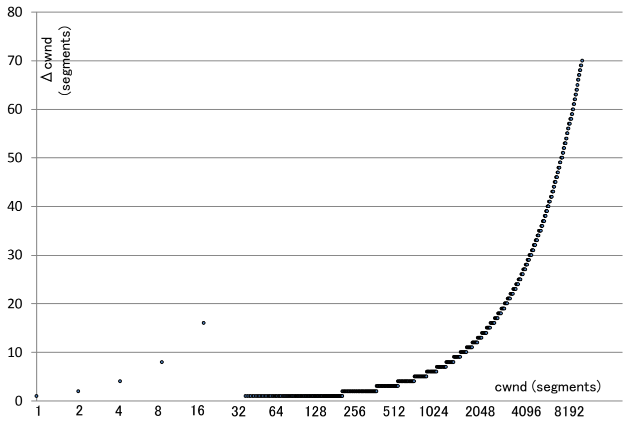


Figure 6. Applying to HS TCP.

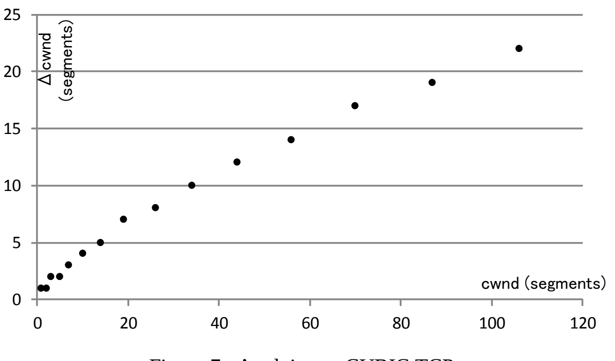


Figure 7. Applying to CUBIC TCP.

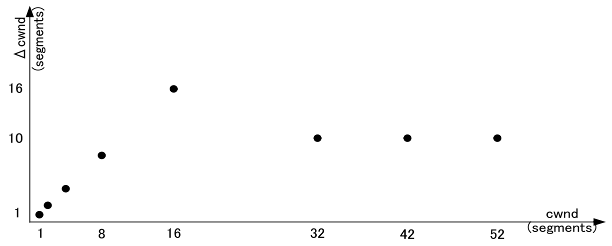


Figure 8. Applying to TCP Illinois.

that the slow start continues until *cwnd* is 16, and that the queuing delay is small in the beginning of the congestion avoidance.

IV. INFERRING IPHONE 5 TCP ALGORITHM

As an example of the inferring of TCP congestion control algorithm using our scheme, we performed an experiment estimating the TCP algorithm of iPhone 5. Fig. 9 shows the configuration. An ftp application on an iPhone 5 terminal communicates with an ftp server through an LTE network and the Internet. While the iPhone 5 uploads a file to the server, it moves on a local train in Tokyo. The packet traces are collected in a PC connected with the iPhone through the remote virtual interface [17].

We collected two packet traces. Fig. 10 shows the results of the first example. Fig. 10 (a) and (b) show the TCP sequence number versus time and *cwnd* value versus time in this communication, respectively. These graph show that the handover happened two times around at 7 second and 35 second, and accordingly, the *cwnd* value decreases. The graph (c) shows the relationship between Δ*cwnd* and *cwnd*. It is noted that the decreases of *cwnd* are not described in this figure. The graph shows the slow start like behavior from *cwnd* =1 to *cwnd* = 23, in which Δ*cwnd* is proportional to *cwnd*. On the other hand, from *cwnd* = 30 to 111, most of observed values for Δ*cwnd* is equal to one. It can be said that the graph in Fig. 10 (c) is quite similar with that in Fig. 3 and so the results of this example says that the TCP congestion control algorithm used in iPhone 5 is AIMD.

Fig. 11 shows another example for iPhone 5. In this example, packet losses occur at 20, 30, 70 and 95 second in the communication, and accordingly the *cwnd* value changes as shown in Fig. 11 (b). Based on this graph, we depicted the relationship between Δ*cwnd* and *cwnd* as shown in Fig. 11 (c). This figure has a proportional part and one segment part similarly with Fig. 10 (c). This result also says that the the TCP congestion control algorithm used in iPhone 5 is AIMD.

V. CONCLUSIONS

This paper proposed a simple but effective scheme inferring the TCP congestion control algorithm from passively collected packet traces. The proposed scheme estimates cwnd values at every RTT intervals from packet traces and makes the correlation beween the *cwnd* values and the differences consecutive *cwnd* values by plotting these values. We showed that the result plotting can explicitly distinguish AIMD, TCP Vegas, TCP Veno, HS TCP, CUBIC TCP and TCP Illinois. As an example, we applied our scheme to identify the TCP congestion control algorithm

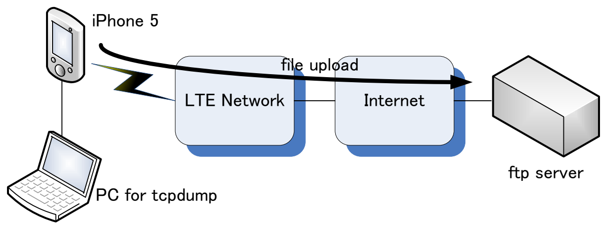
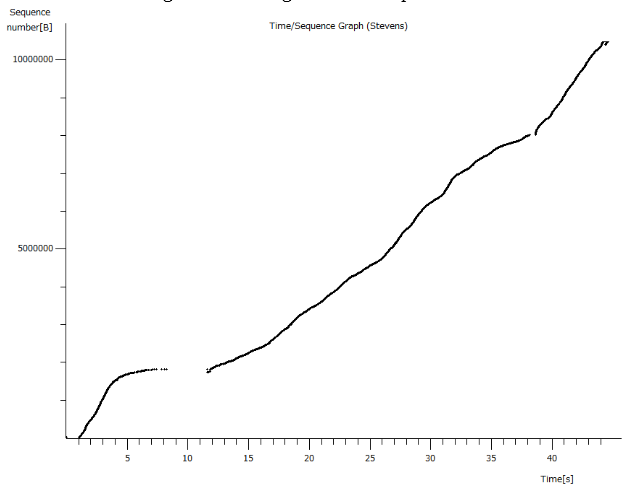
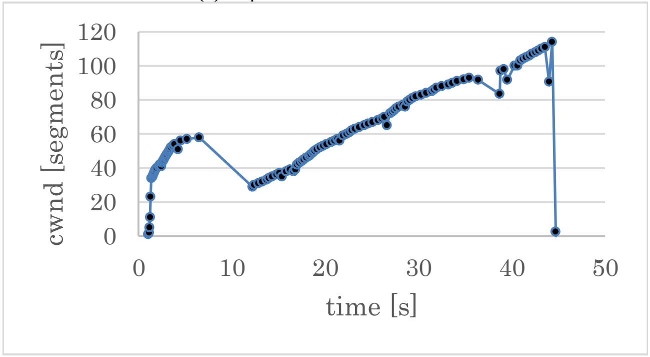


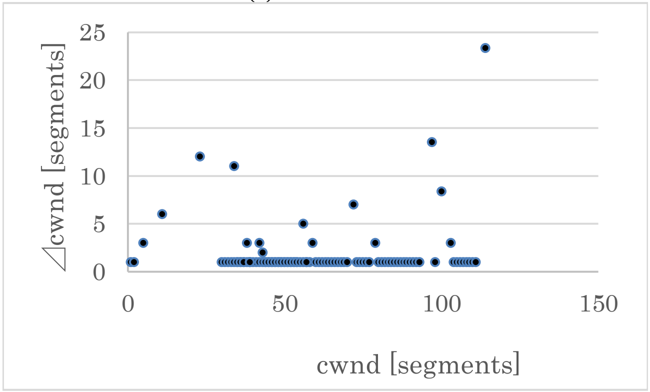
Figure 9. Configuration of experiment.



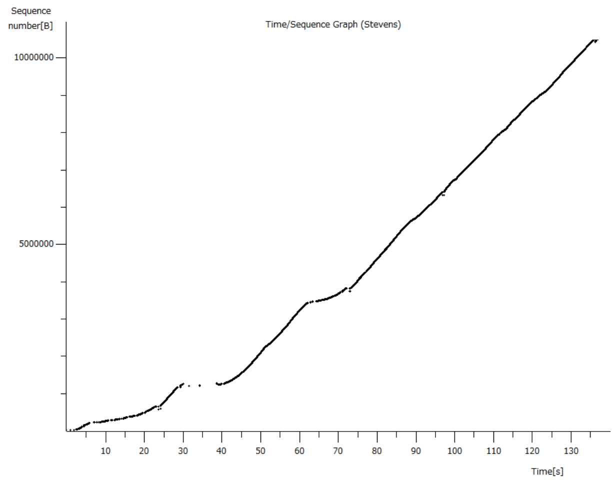
(a) sequence number vs. time.



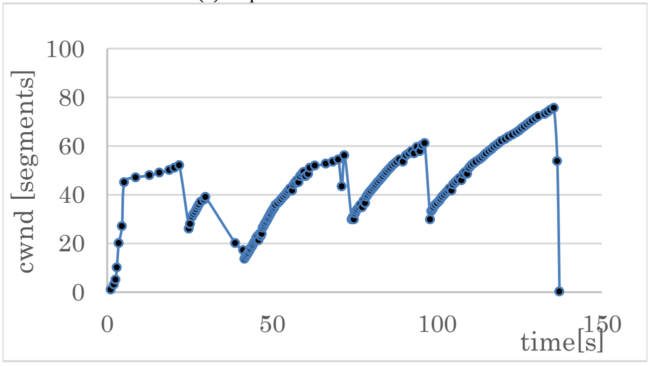
(b) cwnd vs. time.



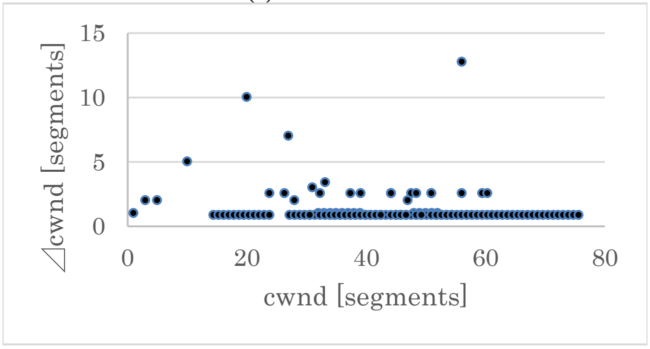
(c) Δcwnd vs. cwnd.
Figure 10. Results of first example.



(a) sequence number vs. time.



(b) cwnd vs. time.



(c) Δcwnd vs. cwnd.

Figure 11. Results of second example.

used in iPhone 5. From two packet traces in which an iPhone 5 terminal is sending ftp data, our scheme showed two graphs showing the AIMD like relationship between $\Delta cwnd$ and $cwnd$. We could successfully conclude that the algorithm used in iPhone 5 is AIMD from these results.

REFERENCES

[1] V. Javobson, "Congestion Avoidance and Control," ACM SIGCOMM Comp. Commun. Review, vol. 18, no. 4, Aug. 1988, pp. 314-329.

[2] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm," IETF RFC 3728, April 2004.

[3] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-Host Congestion Control for TCP," IEEE Commun. Surveys & Tutorials, vol. 12, no. 3, 2010, pp. 304-342.

[4] S. Floyd, "HighSpeed TCP for Large Congestion Windows," IETF RFC 3649, Dec. 2003.

[5] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," ACM SIGOPS Operating Systems Review, vol. 42, no. 5, July 2008, pp. 64-74.

[6] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," Proc. ACM MobiCom '01, July 2001, pp. 287-297.

[7] L. Brakmo and L. Perterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," IEEE J. Selected Areas in Commun., vol. 13, no. 8, Oct. 1995, pp. 1465-1480.

[8] D. Wei, C. Jin, S. Low, and S. Hegde, "FAST TCP: Motivation, Architecture, Algorithms, Performance," IEEE/ACM Trans. on Networking, vol. 14, no. 6, Dec. 2006, pp. 1246-1259.

[9] C. Fu and S. Liew, "TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks," IEEE J. Selected Areas in Commun., vol. 21, no. 2, Feb. 2003, pp. 216-228.

[10] S. Liu, T. Bassar, and R. Srikant, "TCP-Illinois: A loss and delay-based congestion control algorithm for high-speed networks," Proc. VALUETOOLS '06, Oct. 2006.

[11] V. Paxson, "Automated Packet Trace Analysis of TCP Implementations," ACM Comp. Commun. Review, vol. 27, no. 4, Oct. 1997, pp.167-179.

[12] S. Jaiswel, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Inferring TCP Connection Characteristics Through Passive Measurements," Proc. INFOCOM 2004, March 2004, 1582-1592.

[13] J. Oshio, S. Ata, and I. Oka, "Identification of Different TCP Versions Based on Cluster Analysis," Proc. ICCCN 2009, Aug. 2009, pp. 1-6.

[14] F, Qian, A. Gerber, and Z. Mao, "TCP Revisited: A Fresh Look at TCP in the Wild," Proc. IMC '09, Nov. 2009, pp. 76-89.

[15] J.Padhye and S. Floyd, "On inferring TCP behavior," Proc. ACM SIGCOMM, Aug. 2001, pp.287-298.

[16] P. Yang, W. Luo, L. Xu, J. Deogun, and Y. Lu, "TCP Congestion Avoidance Algorithm Identification," Proc. ICDCS '11, June 2011, pp. 310-321.

[17] Apple Inc. "Technical Q&A QA 1176 Getting a Tacket Trace," Available from:
https://developer.apple.com/library/ios/qa/qa1176/_index.html#//apple_ref/doc/uid/DTS10001707, 2014.08.13.