Comparing TCP Congestion Control Algorithms Based on Passively Collected Packet Traces

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

Abstract— **Recently, traffic in the Internet increases largely according to the improvement of network capacity. However, it is sometimes pointed out that a small number of giant users exhaust large part of network bandwidth. In order to resolve such problems, a practical way is to suppress large traffic flows which do not conform to Transmission Control Protocol (TCP) congestion control algorithms. For this purpose, the network operators need to infer congestion control algorithms of individual TCP flows using passively monitored packet traces in the middle of networks. On the other hand, a lot of TCP congestion control mechanisms have been introduced recently. Although there are several proposals on inferring them, no schemes are proposed which can analyze recently introduced TCP congestion control algorithms based on the passive approach. This paper proposes a new passive scheme to compare most of recently proposed congestion control algorithms. It estimates the congestion window size (*cwnd*) at a TCP sender at round-trip time intervals, and specifies the *cwnd* growth as a function of the estimated value of *cwnd* and the *cwnd* decrease parameter at individual congestion events. This paper shows the results of applying our scheme to eight congestion control algorithms and shows that they can be identified from passively monitored traces.**

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

I. INTRODUCTION

The TCP congestion control [1] is a mechanism for a data sender to limit its rate of injecting data segments into the network when it is congested. More specifically, a TCP sender transmits data segments under the limitation of the congestion window size (*cwnd*) maintained within the sender side, beside the advertised window reported from a TCP receiver. The value of *cwnd* grows up as a sender receives acknowledgment (ACK) segments and is decreased when it detects congestions. How to grow and decrease *cwnd* is the key of congestion control algorithm.

Since the congestion control came to be used in TCP, only a few algorithms, such as Tahoe, Reno and NewReno [2], were used commonly for a long time. According to the diversification of network environments, however, many TCP congestion control algorithms have emerged [3]. For example, High Speed (HS) TCP [4], CUBIC TCP [5], and Hamilton TCP [6] are designed for high speed and long delay networks. On the other hand, TCP Westwood+ [7] is designed for lossy wireless links. While those algorithms are based on packet losses, TCP Vegas [8] triggers 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 grow *cwnd*.

Recently, the traffic in the Internet increases largely according to the improvement of network capacity. However, it is sometimes pointed out that a small number of giant users exhaust large part of network bandwidth. Since most of traffic in the Internet uses TCP, the network congestions will be resolved by the TCP congestion control mechanisms. However, if any giant users do not conform to those mechanisms, the problem will be worse. So, an important approach for network operators is to infer congestion control algorithm using passively monitored packet traces and to discriminate TCP unfriendly traffic flows.

This type of TCP congestion control inferring is called a passive approach. It has some limitations in the testing ability because it needs to use packet traces as they are, but is non-intrusive and can be applied to any link in the Internet if the traffic over the link can be monitored. So far, several studies are proposed for passive approaches [11]-[14]. However, there are no proposals on inferring the recently introduced algorithms, in the contrast with the active approach, where an active tester sends test inputs to a target node and checks the replies [15].

In our former paper [16], we presented a new scheme on the passive TCP congestion control algorithm inferring, which is a basis of this paper. However, the paper has some problems in the sense that it focused only on the *cwnd* growth function and that it applied the idea only to a packet trace using TCP Reno/NewReno.

In this paper, we propose a complete scheme to compare the TCP congestion control algorithms. The scheme focuses on not only the *cwnd* growth function, as in our former paper, but also the decrease parameter at the congestion detection. This paper also applies our scheme to most of recently proposed congestion control algorithms implemented in the Linux operating system, with the experimental results verifying our scheme through actually collected packet traces.

The rest of this paper consists of the following sections. Section 2 surveys the related works. Section 3 proposes our scheme. Section 4 gives the results that our scheme is applied to congestion control algorithms actually. In the end, Section 5 gives the conclusions of this paper.

 ISBN: 978-1-61208-439-8

II. Related Works

In the traditional methods [11][12] of the passive approach, a TCP sender's state machine is estimated from packet traces and compared with the behaviors of known algorithms, and the most likely algorithm is selected. These methods need complicated logic and are only applied to early stage algorithms, such as Tahoe, Reno and NewReno. Oshio et al. [13] estimates the changes of *cwnd* values and extracts characteristics, such as the ratio of *cwnd* increased by one. Based on these characteristics, it discriminates one of two different versions randomly selected out of fourteen 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 for detecting the irregular retransmissions, and the extraction of flow clock to find TCP data transmissions controlled by the application or link layer factors. As an example of the active approach, Yang et al. [15] proposes the scheme to actively identify the TCP algorithm of a remote web server. It makes a web server send 512 data segments under the controlled network environment and observes the number of data segments contiguously transmitted without receiving any ACK segments. It then estimates the window growth function and the decrease parameter, and using those estimations, determines the TCP algorithm out of all default TCP algorithms and most non-default TCP algorithms of major operating system families.

III. Proposal

A. Design Principle

A TCP congestion control algorithm can be described by the following two characteristics.

- The window growth function, which determines how an algorithm grows *cwnd* while there is no congestion.
- The multiplicative decrease parameter (denoted by β), which determines the slow start threshold (*ssthresh*) such that

$$ssthresh = cwnd \text{ just before congestion} \times (1 - \beta)$$

The goal of our scheme is to compare TCP congestion control algorithms by specifying those two characteristics using only packet traces collected passively.

The window growth function is defined differently by individual TCP congestion control algorithms. For example, TCP Reno/NewReno defines it as a behavior when a sender receives a new ACK segment. On the other hand, CUBIC TCP defines it as a function of the elapsed time from the last window reduction. For the purpose of our scheme, however, the window growth function needs to be specified in the same framework for different congestion control mechanisms. We have decided to specify it as a function of *cwnd* values estimated at RTT intervals [16].

The multiplicative decrease parameter can be identified from the sequence of estimated *cwnd* values by detecting fast retransmit events.

B. Estimating cwnd Values at RTT Intervals

In the passive approach, packet traces are collected at some monitoring point in the network. So, the time associated with a packet is not the exact time when the node focused sends/receives the packet. Our scheme adopts the following approach to estimate *cwnd* values at RTT intervals using the TCP time stamp option.

- Pick up an ACK segment in a packet trace. Denote this ACK segment by *ACK1*.
- Search for the data segment whose TSecr (time stamp echo reply) is equal to TSval (time stamp value) of *ACK1*. Denote this data segment by *Data1*.
- Search for the ACK segment which acknowledges *Data1* for the first time. Denote this ACK segment by *ACK2*. Denote the ACK segment prior to *ACK2* by *ACK1'*
- Search for the data segment whose TSecr is equal to TSval of *ACK2*. Denote this data segment by *Data2*.

From this result, we estimate a *cwnd* value at the timing of receiving *ACK1* as in (1).

$$cwnd = \left\lfloor \frac{seq \text{ in } Data2 - ack \text{ in } ACK1'}{MSS} \right\rfloor \text{ (segments)} \qquad (1)$$

Here, *seq* means the sequence number, *ack* means the acknowledgment number of TCP header, and *MSS* is the maximum segment size. $\lfloor a \rfloor$ is the truncation of a.

C. Specifying Window Growth Function

Using the sequence of *cwnd* values obtained above, our scheme specifies the window growth function of a focused TCP communication in the following way [16].

- Plot *cwnd* values at RTT intervals in relation to the time associated with the value.
- Select a portion of the *cwnd* vs. time graph where *cwnd* is growing up continuously.
- Compute the difference of adjacent *cwnd* values (denote it by $\Delta cwnd$) for the selected portion, and plot $\Delta cwnd$ versus *cwnd*.

The $\Delta cwnd$ vs. *cwnd* graph obtained here is considered as a representation of the window growth function. As described in the next section, the derived function will show characteristics which can distinguish an individual congestion control mechanism from others.

D. Specifying Multiplicative Decrease Paremeter

Our scheme specifies the multiplicative decrease parameter in the following way.

- From the *cwnd* vs. time graph, select fast retransmit events by identifying portions where *cwnd* drops to some value other than one segment.
- Examine the *cwnd* values just before and just after the drop.

 ISBN: 978-1-61208-439-8

- Compute $1 - \frac{cwnd\ after\ the\ drop}{cwnd\ before\ the\ drop}$ and use it as an estimation of β.

IV. Applying Proposal to Various TCPs

In this section, we show the expected features of individual congestion control algorithms identified by our scheme, and results of experiments applied to actual packet traces.

A. Experiment Conditions

In the experiment, sending and receiving terminals are connected via a bridge. The bridge inserts 100 msec delay (50 msec in one way) and packet losses whose probability is 1.0×10^{-4}. These values are selected for emulating an wide area Internet communication. The sending terminal and the bridge are connected by a 100 Mbps Ethernet link. The receiving terminal and the bridge are connected by an Ethernet link or an IEEE 802.11g WLAN. The data sending is performed by iperf, and is monitored by tcpdump at the sender. We used either result of an Ethernet link or a WLAN depending on individual algorithms.

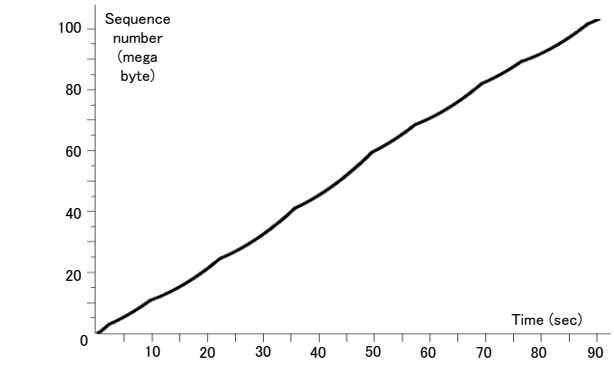
B. Applying to TCP Reno/NewReno

In TCP Reno/NewReno, *cnwd* (in unit of segment) grows up, for a new ACK segment, by one in the slow start phase and by 1/*cwnd* in the congestion avoidance phase. By considering the possibility that the delayed ACK is used, the growth of *cwnd* during a RTT will be $cwnd/2 \leq \Delta cwnd \leq cwnd$ in the slow start phase, and $\Delta cwnd = 0\ or\ 1$ in the congestion avoidance phase. As for the multiplicative decrease parameter, $\beta = 0.5$.

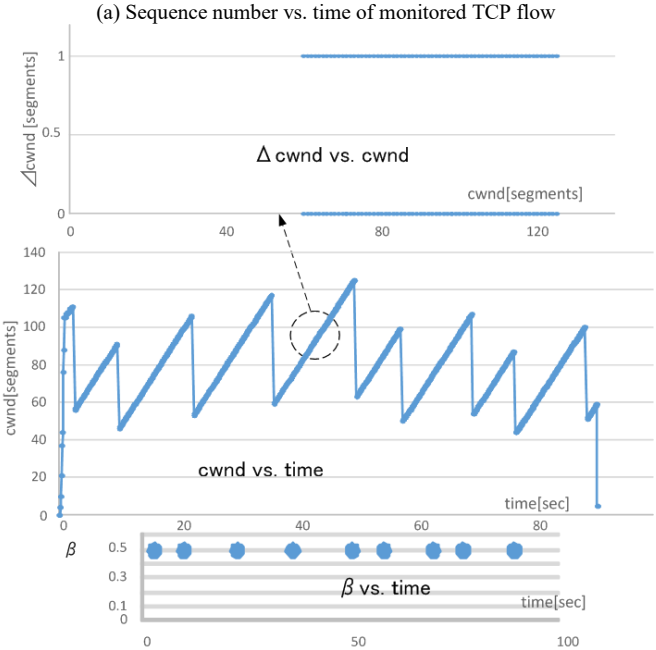
Figure 1 shows experimental results for TCP Reno/ NewReno. In Figure 1(a), the change of sequence number is shown along the time sent from the TCP sender. This figure corresponds to the information included in the packet trace. From this result, the sequence of *cwnd* values at RTT intervals are computed by the algorithm described in II.B, which is given in the *cwnd* vs. time graph in (b) of this figure. In this graph, the portion marked by a circle is selected, and the *Δcwnd* vs. *cwnd* graph is plotted. The dropping portions in the *cwnd* vs. time graph generate the β vs. time graph. These two graphs give the features expected above. It should be noted that the ratio of *Δcwnd* = 0 and 1 is 1:1. This is reasonable because the delayed ACK sends an ACK segment for every other data segment and, therefore, *Δcwnd* will be one every other RTT interval.

C. Applying to HS TCP

HS TCP is designed to obtain high throughput over wide bandwidth and long delay networks. It grows *cwnd* to $cwnd + {a(cwnd)}/{cwnd}$ in response to every new ACK segment, and decrease *cwnd* to $(1 - b(cwnd)) \times cwnd$ at a congestion event. That is, it changes the increase and decrease parameters, *a*(*cwnd*) and *b*(*cwnd*), depending on *cwnd* value. More specifically, *a*(*) and *b*(*) are defined as follows.



(a) Sequence number vs. time of monitored TCP flow



(b) Estimated cwnd increasing function and decrease parameter

Figure 1. Experimental results for TCP Reno/NewReno (using Ethernet link).

$$a(cwnd) = \frac{0.156 \times cwnd^{0.8} \times b(cwnd)}{2 - b(cwnd)} \quad (2)$$

$$b(cwnd) = (0.1 - 0.5) \times \frac{\log cwnd - \log 38}{\log 83000 - \log 38} + 0.5 \quad (3)$$

From those equations, when *cwnd* is 38, 118, or 221, *a*(*cwnd*) is 1, 2, or 3 segments and *b*(*cwnd*) is 0.50, 0.44, or 0.41, respectively. Considering that the passive approach can only detect the *cwnd* value in the unit of segment and that there is a case the delayed ACK is used, the estimated *Δcwnd* will be as follows.

$$\Delta cwnd = \begin{cases} 0\ or\ 1\ (cwnd < 38) \\ 1\ or\ 2\ \ (38 \leq cwnd < 118) \\ 1, 2\ or\ 3\ \ (118 \leq cwnd < 221) \end{cases} \quad (4)$$

On the other hand, the estimated value of β will be the same as *b*(*cwnd*).

Figure 2 shows experimental results for HS TCP. It shows only the graphs obtained in our proposal. From the

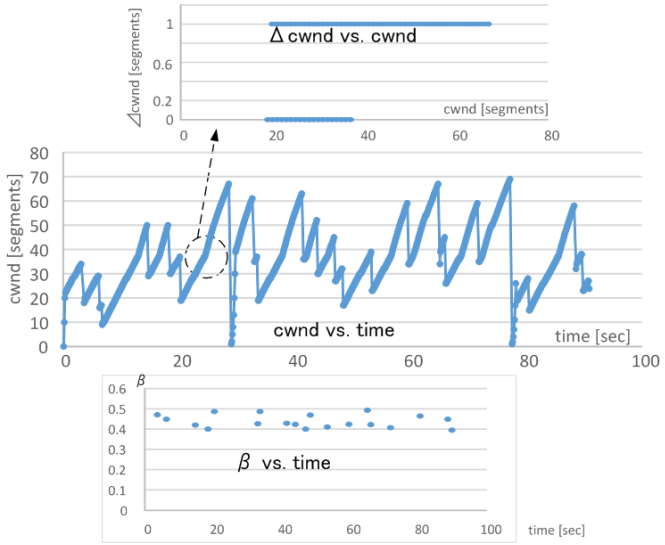


Figure 2. Experimental results for HS TCP (using WLAN).

Δcwnd vs. *cwnd* graph, *Δcwnd* is 0 or 1 and their ratio is 1:1 when *cwnd* <38. When *cwnd* ≥ 38, *Δcwnd* is 1. This result is consistent with the expectation above, and especially it should be noted that the *Δcwnd* value changes at the *cwnd* value of 38. As for the multiplicative decrease parameter, β is between 0.4 and 0.5 and this result is also consistent with the expectation.

D. Applying 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 the decrease parameter, and $cwnd_{max}$ is the value of *cwnd* just before the loss detection in the last congestion event. We approximate *Δcwnd* by $RTT \times \frac{d(cwnd)}{dT}$ and obtain (6) by representing it in *cwnd* [16].

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

The decrease parameter is defined by $\beta = 0.2$ in the original CUBIC. It is 0.3 in the new versions of CUBIC TCP [3].

Figure 3 shows experimental results for CUBIC TCP. The curve in the *Δcwnd* vs. *cwnd* graph has two characteristics. One is that it follows a $\sqrt[3]{x^2}$ curve and the other is that it has parts in both sides of a point of *Δcwnd* = 0. So, it is considered that this result is consistent with (6). As for the decrease parameter, the result is $\beta \approx 0.3$ and this means that the used CUBIC software is a new version.

E. Applying to Hamilton TCP

Hamilton TCP is another example that defines *cwnd* as a function of a time. It defines the increase parameter *a* of *cwnd*, similar with that of HS TCP, as a function of elapsed time *T* since the last congestion event in the following way.

$$a(T) = \begin{cases} 1 + 10(T - T_{low}) + 0.25(T - T_{low})^2 & (T \geq T_{low}) \\ 1 & (T < T_{low}) \end{cases} \qquad (7)$$

Here, T_{low} is a threshold for switching the low-speed mode and the high-speed mode. *a*(*T*) is an increase of *cwnd* during a RTT interval, we can obtain an approximate value of *cwnd* by integrating (7). First of all, we compute the square completion the upper equation of (7), and obtain (8).

$$\Delta cwnd = \frac{1}{4}(T - T_{low} + 20)^2 - 99 \qquad (8)$$

By integrating (8) and substituting *Δcwnd*, *cwnd* is computed as a function of *Δcwnd* in the following way.

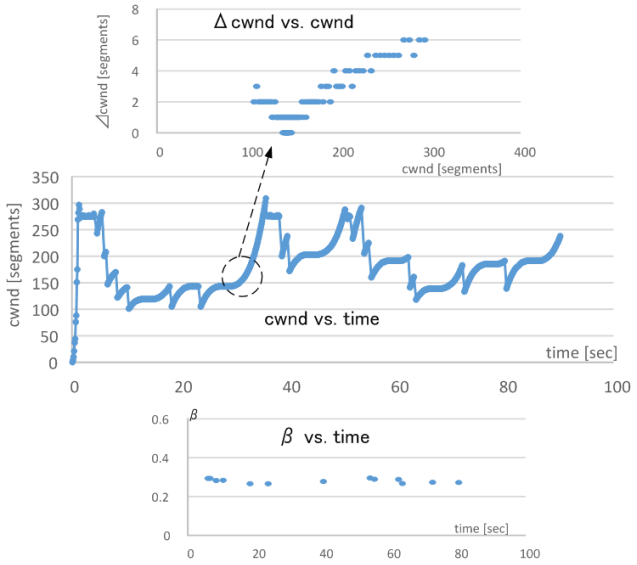


Figure 3. Experimental results for CUBIC TCP (using Ethernet link).

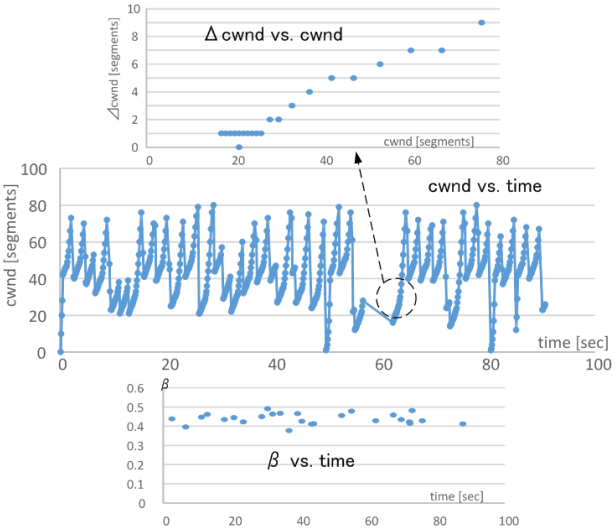


Figure 4. Experimental results for Hamilton TCP (using WLAN).

$$cwnd = \frac{1}{3RTT}\left(\sqrt{\Delta cwnd + 99}\right)^3 - \frac{198}{RTT}\sqrt{\Delta cwnd + 99} + C \quad (9)$$

Here, C is a constant. This result means that *cwnd* is a function of $\Delta \text{cwnd}^{\frac{3}{2}}$. So it is considered that, by computing the inverse function, *Δcwnd* will be represented by a function of $\text{cwnd}^{\frac{2}{3}}$. This is a similar result with CUBIC TCP. But, in the case of Hamilton TCP, the TCP Reno part exists before a $\sqrt[3]{x^2}$ curve, and there in only an increasing part unlike CUBIC TCP. As for the multiplicative decrease parameter, $\beta = 0.5$ is expected.

Figure 4 shows experimental results for Hamilton TCP. The curve in the *cwnd* vs. *Δcwnd* graph presents the exact characteristics described above. As for β, the result value is between 0.4 and 0.5, which is acceptable for the estimation.

F. Applying to TCP Westwood+

TCP Westwood+ is based on the end-to-end bandwidth estimate using the rate of acknowledged data in returning ACK segments. Its congestion control is triggered by packet losses. While there are no packet losses, it increases *cwnd* by the same algorithm with TCP Reno for every new ACK segment. At the same time, the estimated bandwidth (b_k) is computed every RTT in the following way.

$$b_k = d_k / \Delta_k \quad (10)$$

Here, d_k is the amount of data acknowledged during the last RTT (Δ_k). The measured value b_k is applied to an exponential moving average filter and the averaged bandwidth estimation (BWE_k) is obtained.

$$BWE_k = 0.9 \times BWE_{k-1} + 0.1 \times b_k \quad (11)$$

When three duplicate ACKs are received, *cwnd* is set to the value of $BWE \times RTT_{min}/MSS$. That is, *cwnd* is decreased to a specific value not using a multiplicative decrease parameter. From those definitions, the expectation of *Δcwnd* will be 0 or 1, which is the same with TCP Reno. The expectation of β will be as in (12).

$$1 - \frac{BWE \times RTT_{min}}{MSS \times cwnd_{max}} \quad (12)$$

Here, $cwnd_{max}$ is the value of *cwnd* just before the last loss detection.

Figure 5 shows experimental results for TCP Westwood+. In the *Δcwnd* vs. *cwnd* graph, *Δcwnd* takes 1 and 0, and its ratio is 1:1. This is the same with TCP Reno and conforms to the expectation. On the other hand, in the β vs. time graph, β takes various values between 0.2 and 0.5. Basically, β itself has no meaning in this case, and in this sense the results conform to the expectation.

G. Applying to TCP Vegas

TCP Vegas estimates the bottleneck buffer size using the current values of *cwnd* and RTT, and the minimal RTT for the TCP connection, according to (13).

$$BufferSize = cwnd \times \frac{RTT - RTT_{min}}{RTT} \quad (13)$$

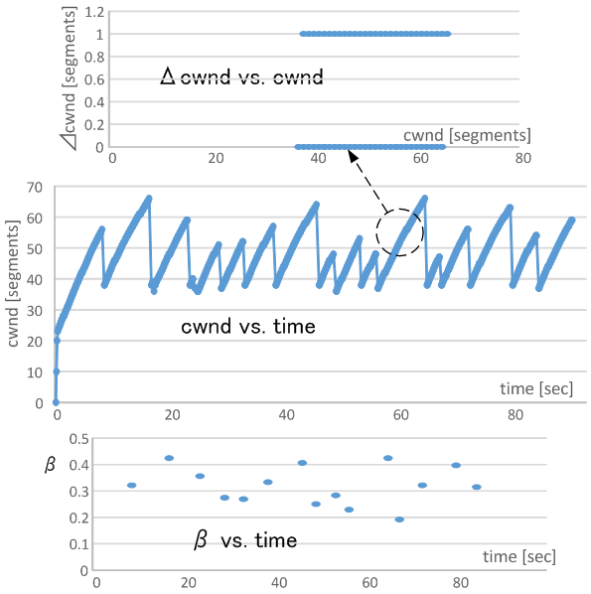


Figure 5. Experimental results for TCP Westwood+ (using WLAN).

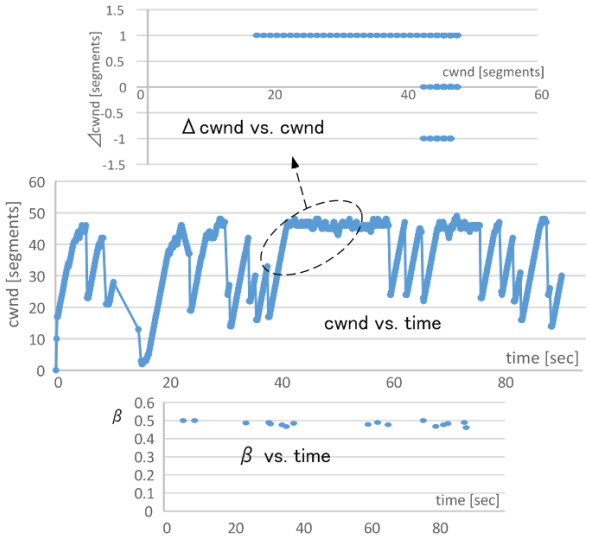


Figure 6. Experimental results for TCP Vegas (using WLAN).

At every RTT interval, Vegas uses this *BufferSize* to control *cwnd* in the congestion avoidance phase in the following way.

$$\Delta cwnd = \begin{cases} 1 & (BufferSize < A) \\ 0 & (A \leqq BufferSize \leqq B) \\ -1 & (BufferSize > B) \end{cases} \quad (14)$$

Here, A = 2 and B = 4 (in unit of segment) are used in the Linux operating system. The decrease parameter is $\beta = 0.5$.

Figure 6 shows the results for TCP Vegas. In the *Δcwnd* vs. *cwnd* graph, *Δcwnd* is 1 while *cwnd* is below 40, which corresponds to the part of increasing *cwnd*. After that, around *cwnd* is 45, the situations that *Δcwnd* is 0, 1 and -1 are mixed. This result conforms to the expectation above. In the β vs. time graph, $\beta = 0.5$, which matches the expectation.

 ISBN: 978-1-61208-439-8

H. Applying to TCP Veno

TCP Veno (Vegas and ReNO) uses the *BufferSize* in (13) to adjust the growth of *cwnd* in the congestion avoidance phase as follows. If BufferSize > B (*B* is the Vegas parameter *B*), cwnd grows by 1/cwnd for every other new ACK segment, and otherwise, it grows in the same manner with TCP Reno. Therefore, if the delayed ACK is not used, *Δcwnd* at RTT intervals will be as in (15).

$$\Delta cwnd = \begin{cases} 1 \text{ or } 0 (BufferSize > B) \\ 1 \ (BufferSize \leqq B) \end{cases} \quad (15)$$

If the delayed ACK is used, $\Delta cwnd = 0 \text{ or } 1$ even if $BufferSize \leqq B$. But in this case, the ratio of *Δcwnd* being 1 and 0 is different for *BufferSize*. It will be 1:3 for $BufferSize > B$, and 1:1 for $BufferSize \leqq B$. The multiplicative decrease parameter is defined as in (16).

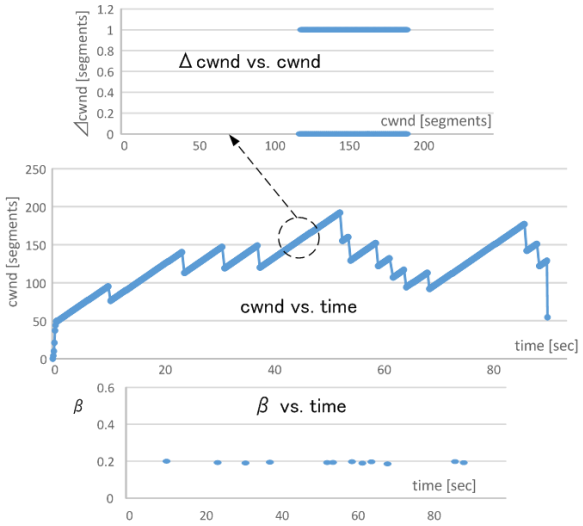


Figure 7. Experimental results for TCP Veno (using Ethernet link).

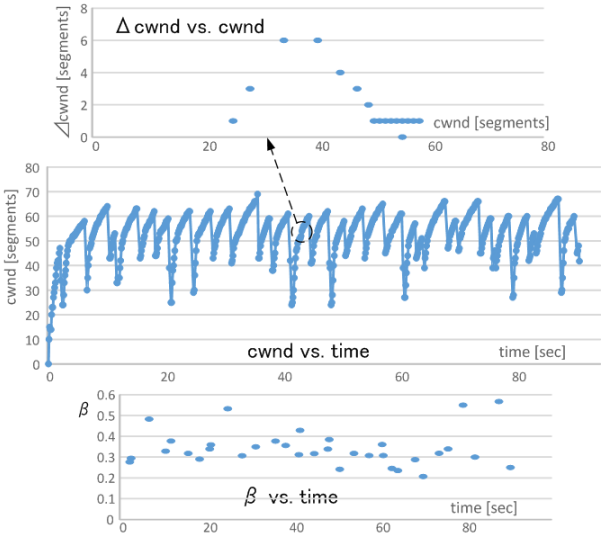


Figure 8. Experimental results for TCP Illinois (using WLAN)

$$\beta = \begin{cases} 0.5 \ (BufferSize > B) \\ 0.2 \ (BufferSize \leqq B) \end{cases} \quad (16)$$

Figure 7 shows experimental results for TCP Veno. In the *cwnd* vs. *Δcwnd* graph, *Δcwnd* takes 1 and 0, but its ratio is 1:1. On the other hand, the β vs. time graph shows that $\beta = 0.2$. These results are consistent with the expectation when *BufferSize* is less than and equal to *B*.

I. Applying to TCP Illinois

TCP Illinois changes the increase parameter, $a(Q)$, and the decrease parameter, $b(Q)$, of *cwnd*, which are similar with those of HS TCP, according to the queuing delay, *Q*. The queuing delay is measured by the increase of RTT from the minimum RTT for a TCP connection. In the Linux operating system, $a(Q)$ changes from 0.1 to 10 in unit of segment. $b(Q)$ changes from 0.125 to 0.5. Those values are updated once per every RTT. In the expectation, *Δcwnd* will be defined by $\frac{1}{2}a(Q) \leq \Delta cwnd \leq a(Q)$ and β will be $b(Q)$.

Figure 8 shows experimental results for TCP Illinois. In the *Δcwnd* vs. *cwnd* graph, *Δcwnd* increases from 1 to 6 and then decreases to 1 again. This will reflect the delay in the communication. The β vs. time graph, β has the values between 0.2 and 0.6. These conform to the expectations.

V. Conclusions

This paper presented that the TCP congestion control algorithms can be characterized from only passively collected packet traces, by specifying the *cwnd* growth function as *Δcwnd* vs. *cwnd*, and the multiplicative decrease parameter. We applied our scheme to Reno/NewReno, HS TCP, CUBIC, Hamilton, Westwood+, Vegas, Veno and Illinois, and indicated that individual algorithms show characteristics which can identify the individuals from others. Our future works include identifying congestion control algorithms automatically and inferring from packet traces which contain only one way TCP packet traces.

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] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long distance networks," Proc. Int. Workshop on PFLDnet, Feb. 2004, pp. 1-16.

[7] L. Grieco and S. Mascolo, "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control," ACM Computer Communication Review, vol. 34, no. 2, April 2004, pp. 25-38.

 ISBN: 978-1-61208-439-8

[8] 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.

[9] C. Fu and S. Liew, “TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks,” IEEE J. Sel. 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, pp. 1-13.

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

[16] T. Kato, A. Oda, S. Ayukawa, C. Wu, and S. Ohzahata, “Inferring TCP Congestion Control Algorithms by Correlating Congestion Window Sizes and their Differences,” Proc. IARIA ICSNC 2014, Oct. 2014, pp.42-47.

 ISBN: 978-1-61208-439-8