A Study on How to Characterize TCP Congestion Control Algorithms from Unidirectional Packet Traces

Toshihiko Kato, Leelianou Yongxialee, Ryo Yamamoto, and Satoshi Ohzahata
Graduate School of Information Systems
University of Electro-Communications
Tokyo, Japan
e-mail: kato@is.uec.ac.jp, zoosiab@net.is.uec.ac.jp, ryo_yamamoto@is.uec.ac.jp, ohzahata@is.uec.ac.jp

Abstract— Although traffic in the Internet increases largely, 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 packet traces collected passively in the middle of networks. We proposed, in our previous paper, a new scheme to characterize TCP algorithms from packet traces. 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*. We showed that our previous scheme can characterize most TCP algorithms introduced recently. In an actual network environment, however, a packet trace captured over some link, especially a backbone link, often contains only unidirectional TCP segments due to the asymmetric routing. In this case, it is difficult to estimate the *cwnd* itself, and a new analysis scheme is required. This paper shows a study on how to characterize the TCP congestion control algorithms from unidirectional packet traces. We use a data size transmitted during a short period of time and, using it, we apply our former scheme to the unidirectional trace. This paper shows the results that we apply the proposed method to popular TCP algorithms, such as TCP Reno and CUBIC TCP.

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

I. INTRODUCTION

Recently, traffic in the Internet increases largely according to the increase of network capacity. The benefit of this capacity increase needs to be given equally to individual users. However, it is sometimes pointed out that some 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 them intentionally, the problem will be worse. So, an important approach for network operators is to infer congestion control algorithms based on TCP segment exchanges captured over some link in the network. This is called the *passive approach* for TCP congestion control inferring. This is in contrast with the *active approach*, where an active tester sends test sequences to a target node and checks the replies

In the TCP congestion control [1], a data sender transmits data segments under the limitation of the congestion window size (*cwnd*) maintained within the sender itself, beside the advertised window reported from a data 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.

Although there were only a few congestion control algorithms, such as Tahoe, Reno and NewReno [2] in the early stage, many TCP congestion control algorithms have emerged recently [3]. For example, CUBIC TCP [4] and High Speed (HS) TCP [5] are designed for high speed and long delay networks. Among them, CUBIC TCP is used as a standard version in the Linux operating system. While many algorithms are based on packet losses, TCP Vegas [6] triggers congestion control against an increase of round-trip time (RTT). TCP Veno [7] combines loss based and delay based approaches such that the congestion control is triggered by packet losses but the delay determines how to grow *cwnd*.

This proliferation of algorithms complicates their inference in the passive approach. In our previous paper [8], we proposed a scheme for characterizing TCP congestion control algorithms from passively collected packet traces. As far as we know, this is the only passive approach which can handle most of the major TCP algorithms, differently from other proposals [9]-[14].

Our previous proposal requires both TCP data and acknowledgment segments are captured in a packet trace. It observes a RTT by mapping a data segment and its corresponding ACK segment, and estimates the value of *cwnd* as an outstanding data size during the RTT period. However, in an actual network environment, due to the asymmetric routing, a packet trace captured over some link, especially a backbone link, often contains only unidirectional TCP segments. In this case, it is difficult to estimate *cwnd*, and therefore, a new analysis scheme is required.

In this paper, we show a study on how to characterize the TCP congestion control algorithms from unidirectional packet traces. We propose an approach *based on sent data size during a short time period*. From the unidirectional trace, we cannot estimate either the RTT or *cwnd* values. Instead, we presume that the data size sent in a short time period is proportional to a *cwnd* value. Using this data size, we applied our former proposal to the unidirectional trace. We apply the proposed method to TCP Reno, CUBIC TCP, HS TCP, TCP Vegas, and TCP Veno.

The rest of this paper consists of the following sections. Section 2 surveys the related works. Section 3 discusses the detailed study on our proposal through applying the actual packet traces of TCP Reno and CUBIC TCP. Section 4 shows

 ISBN: 978-1-61208-475-6

the results of our scheme being applied to other TCP versions. In the end, Section 5 gives the conclusions of this paper.

II. Related Works

The works on the passive approach TCP congestion control algorithm inference in the early stage [9][10] accepted an approach to keep track of the sender's *cwnd* based on the predefined TCP finite state machine. But, they considered only TCP Tahoe, Reno and New Reno and did not handle any of recently introduced algorithms. Oshio et al. [11] proposed a scheme to discriminate one out of two different TCP versions randomly selected from fourteen versions. They adopted an approach to keep track of changes of *cwnd* from a packet trace and to extract several characteristics, such as the ratio of *cwnd* being increased by one packet. But, they assumed that the discriminator knows which two TCP versions are used in the packet trace. Prior to our previous proposal [8], the only study which can infer the TCP algorithms including those introduced recently was a work by Yang et al. [15]. It is an example of the active approach. It makes a web server send 512 data segments under the controlled network environment, and observes the number of data segments contiguously transmitted. From those results, it estimates the window growth function and the decrease parameter to determine the TCP algorithm. All of the proposals so far, including our previous one, based on bidirectional TCP interactions.

On the other hand, there are several works based on unidirectional packet traces [12]-[14]. T-RAT [12] used an approach to separate a unidirectional packet trace into flights, and then infer the TCP state of each flight (e.g., slow start or congestion avoidance). K. Lan and J. Heidemann [13] proposed an approach to examine characteristics of giant TCP flows in four dimensions, i.e., size, duration, rate, and burstiness, along with their correlations. Qian et al. [14] proposed a scheme to extract more detailed statistical features from unidirectional TCP traces. 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 flow clock to find TCP data transmissions controlled by the application or link layer factors. None of them, however, proposed a way to infer TCP algorithms from unidirectional packet traces.

In this paper, we discuss on a scheme to characterize recent TCP algorithms based on unidirectional traces. We use TCP Reno and CUBIC TCP as examples to design the scheme, and apply it to HS TCP, TCP Vegas and Veno.

III. Study by Analysing TCP Reno and CUBIC TCP

A. Proposal

In the rest of this paper, we use unidirectional packet traces which contain only the information on TCP data segments. From such a trace, we obtain a sequence of the time of individual packet capture and the TCP sequence number contained in captured packets for a specific TCP flow. Figure 1 (a) shows an example of such a sequence. This is selected from a CUBIC TCP trace.

time (sec)	tcp.seq (byte)
23.48312	52023289
23.48327	52024737
23.48368	52026185
23.48407	52027633
23.48433	52029081
...	...
23.49136	52072521
23.49146	52073969
23.5222	52075417
23.52236	52076865
...	...
23.59184	52218809
23.59184	52220257
23.6227	52221705
23.62271	52223153
...	...
23.69383	52370849
23.69383	52372297
23.72319	52373745
23.7232	52375193
23.72364	52376641
...	...

(a) capture time and TCP sequence number

time (sec)	sentData (byte)
23.48312	1448
23.48327	2896
23.48368	4344
23.48407	5792
23.48433	7240
...	...
23.49136	50680
23.49146	52128
23.5222	1448
23.52236	2896
...	...
23.59184	144840
23.59184	146288
23.6227	1448
23.62271	2896
...	...
23.69383	150592
23.69383	152040
23.72319	1448
23.7232	2896
23.72364	4344
...	...

(b) sent data size

Figure 1. Example of a unidirectional packet trace.

From this information, we use the following procedures to characterize the TCP algorithms.

- Check the sequence numbers and select retransmissions which can be detected by its decreasing.
- Pick up a portion where the sequence numbers are increasing continuously (no retransmissions occur). Figure 1 (a) is such a portion in a CUBIC TCP trace.
- Select a short time period to analyze the data size sent during this period. In Figure 1 (b), we select 100 msec as the period and apply it to the no retransmission portion given in (a) in this figure. The data size sent in a TCP segment is calculated by the TCP sequence number of the next segment.
- Use the data size sent during a selected time period as *sentData*. We use this as a value proportional to *cwnd*. In the result of Figure 1 (b), 52,128, 146,288, 152,040 (bytes) are those values.
- Calculate the difference of adjacent *sentData*. We denote it by *⊿sentData*. We presume this value is proportional to the increase of *cwnd*. In Figure 1 (b), 94,100 and 5,812 (bytes) are the values.
- Plot *⊿sentData* versus *sentData* graph and consult the result with the relationship obtained by our previous proposal [8]. We suppose that the result is similar with that in [8]. In the case of TCP Reno, for the estimated value of *cwnd* in a RTT interval, its difference *⊿cwnd* has the relationship with *cwnd* such as $\triangle cwnd = 0$ *or* 1 (in unit of packet) independently of the value of *cwnd*. For, CUBIC TCP, the following relationship is resumed for *⊿cwnd* and cwnd.

$$\triangle cwnd = 3RTT \cdot \sqrt[3]{C}\left(\sqrt[3]{cwnd - cwnd_{max}}\right)^2 \quad (1)$$

Here, C is a predefined constant and $cwnd_{max}$ is the value of *cwnd* just before the last loss detection.

B. Study on Scheme Using TCP Reno

Next, we discuss the effectiveness of the proposal above through some experiments using actual packet traces. We use the packet traces collected in our previous study [8]. In our previous experiment, iperf TCP data transfer is performed between sending and receiving terminals. The sending terminal runs the Linux operating system and one of supported TCP versions is selected in the experiment. Those terminals are connected via a bridge, which inserts 100 msec delay (50 msec in one way) and packet losses whose probability is 1.0×10^{-4}. 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 TCP segments transmitted are monitored by tcpdump at the sending terminal. We used either result of an Ethernet link or a WLAN depending on individual algorithms.

The obtained packet traces contain bidirectional TCP segments. From them, we selected only segments from the sending terminal to the receiving terminal, and obtained unidirectional packet traces such as one given in Figure 1 (a).

First, we tried a TCP Reno trace. We picked up a portion at the trace from 22.5 sec to 35.9 sec containing 11,437 segments whose sequence numbers continue to grow up. We selected 100 msec as the short time period for calculating *sentData*. The orange line in Figure 2 shows the relationship between *sentData* and time in this case. As this figure shows, the *sentData* is increasing linearly along with time. This result seems to reflect the behavior of TCP Reno correctly. Figure 3 shows the relationship between *ΔsentData* and *sentData*. The result indicates that *ΔsentData* mainly takes 0 byte and or 1448 bytes, and that it takes -1448 bytes and 2896 bytes sometimes. This result seems to be similar with that in our previous paper using bidirectional packet traces.

But, it needs to be mentioned that the period of 100 msec is equal to the round-trip delay inserted by the bridge in the experiment. So, the result in Figure 3 becomes similar to that in the previous paper.

As another value of time period, we use 200 msec. Figure 2 blue line shows the relationship between sent data and time in this case. The result is linear, but the value is twice of that in the above case. Figure 4 shows the relationship between *ΔsentData* and *sentData* obtained from the result above. In this case, the result shows that *ΔsentData* mainly takes 1448 bytes and or 4344 bytes. These values correspond to the segment size or three times of it. This result comes from the fact that the period for *sentData* calculation differs from the RTT value. We can say that, for TCP Reno, *ΔsentData* keeps constant irrelevant with *sentData* and takes two values mainly.

Figure 5 shows the relationship between *Δcwnd* and *cwnd* obtained in our previous paper in the same portion in the trace. The result in Figure 3 is similar with that in Figure 5. So, our scheme in this paper seems to be able to characterize TCP Reno from unidirectional packet traces. This is because the proposed scheme uses a time period identical to the delay inserted in the experiment. In Figure 4, the time period is 200 msec which is twice of the inserted delay. Still, there is a strong similarity among the result in Figure 4 and that in Figure 5, and so our scheme is working well in the case that the time period for calculating *sentData* is 200 msec.

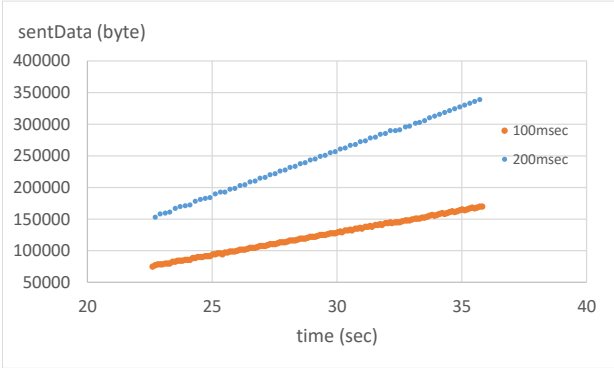

Figure 2. *sentData* vs. time for TCP Reno with 100 and 200 msec period.

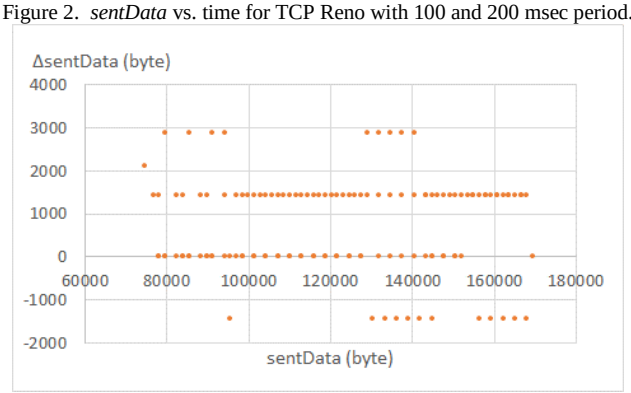

Figure 3. *ΔsentData* vs. *sentData* for TCP Reno with 100 msec period.

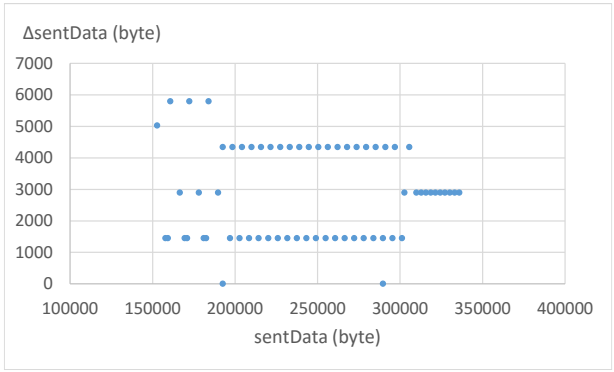

Figure 4. *ΔsentData* vs. *sentData* for TCP Reno with 200 msec period.

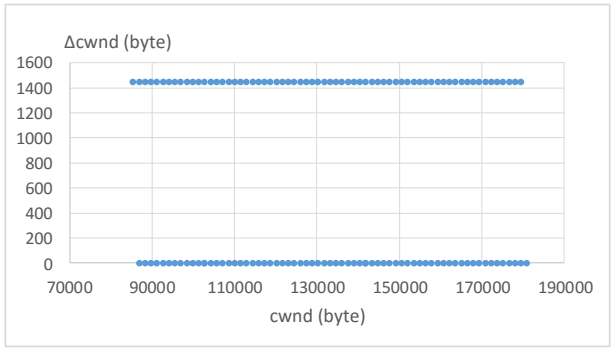

Figure 5. *Δcwnd* vs. *cwnd* for TCP Reno [8].

C. Study on Scheme Using CUBIC TCP

Next, we tried a CUBIC TCP trace as another example. As described above, our previous paper indicates that, in the case of CUBIC TCP, *Δcwnd* has the relationship of the square of cubic root of *cwnd*. In order to examine whether this relationship can be applied to *ΔsentData* and *sentData*, we examined the unidirectional packet trace derived from the trace in our previous paper, with 100 msec and 200 msec time period for calculating *sentData*.

We picked up a portion at the trace from 23.5 sec to 38.3 sec containing 27,041 segments whose sequence numbers continue to grow up. Figure 6 shows the relationship between *sentData* and time for both cases of 100 msec and 200 msec time periods. In both cases, the *sentData* is increasing as a cubic curve of time.

Figures 7 and 8 show the relationship between *ΔsentData* and *sentData* for 100 msec and 200 msec time periods, respectively. Both figures indicate that *ΔsentData* is symmetrical for some value of *sentData* (200,000 and 400,000 bytes in Figures 7 and 8, respectively), and that *ΔsentData* is decreasing if *sentData* is smaller than the value and increasing if larger than the value. This situation is similar to the result of our previous result described in [9]. Especially, the situation is clearer for the case of 200 msec time period.

Figure 9 shows the relationship between *Δcwnd* and *cwnd* obtained in our previous paper in the same portion in the trace. The results in Figures 7, 8 and 9 are similar and so we can say that the proposed scheme here works well for CUBIC TCP.

IV. Apply to Other TCP Versions

In this section, we apply our scheme to other TCP versions. In the following study, we use 200 msec as the time period for calculating *sentData* because we suppose that 200 msec will give smoother results.

A. Result Applied 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. The coefficient *a*(*cwnd*) is defined as 1, 2, and 3 when cwnd is 38, 118, and 221 (in unit of packet). So, in our previous approach, the estimated *Δcwnd* will be as follows.

$$\triangle cwnd = \begin{cases} 0 \text{ or } 1 \ (cwnd < 38) \\ 1 \text{ or } 2 \ \ (38 \le cwnd < 118) \\ 1, 2 \text{ or } 3 \ \ (118 \le cwnd < 221) \end{cases} \quad (2)$$

Similarly with the study in Section 3, we made the unidirectional packet trace, and picked up a portion at the trace from 20.2 sec to 28.8 sec containing 2,749 data segments.

Figure 10 shows the relationship between *Δcwnd* and *cwnd* obtained in our previous paper. Figure 11 shows the relationship between *ΔsentData* and *sentData*. These two figures are for the identical portion in the trace. From the results, it might be difficult to say that the method proposed in this paper can characterize the feature of HS TCP.

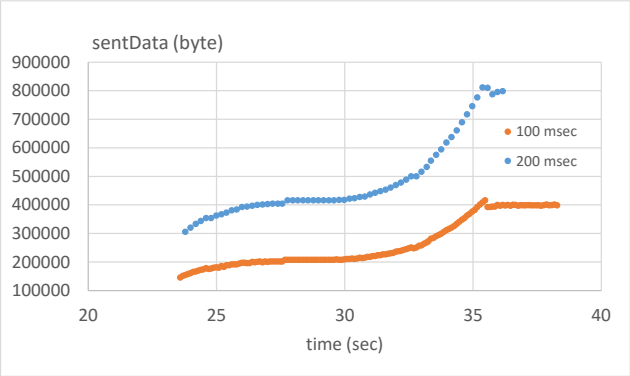

Figure 6. *sentData* vs. time for CUBIC TCP with 100 and 200 msec period.

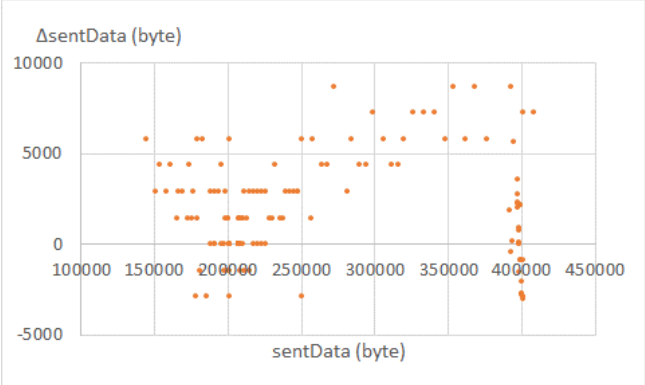

Figure 7. *ΔsentData* vs. *sentData* for CUBIC TCP with 100 msec period.

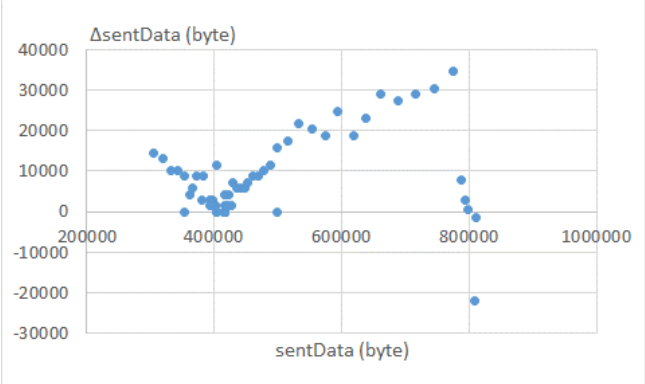

Figure 8. *ΔsentData* vs. *sentData* for CUBIC TCP with 200 msec period.

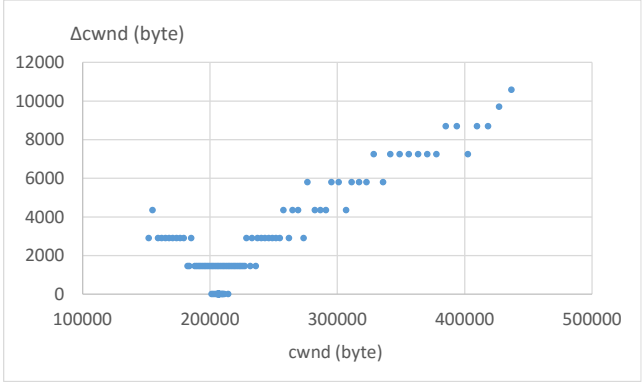

Figure 9. *Δcwnd* vs. *cwnd* for CUBIC TCP [8].

 ISBN: 978-1-61208-475-6

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

$$\Delta cwnd = \begin{cases} 1 & (BufferSize < A) \\ 0 & (A \leqq BufferSize \leqq B) \\ -1 & (BufferSize > B) \end{cases} \quad (3)$$

Here, A = 2 and B = 4 (in unit of segment) are used in the Linux operating system (in unit of packet).

We made the unidirectional packet trace from the trace in the previous experiment, and picked up a portion at the trace from 38.0 sec to 59.7 sec containing 8,155 data segments.

Figure 12 shows the relationship between *Δcwnd* and *cwnd* obtained in our previous paper. Figure 13 shows the relationship between *ΔsentData* and *sentData*. For the values of *sentData* 100,000 through 120,000 bytes in Figure 13, *ΔsentData* are distributed between +10,000 bytes and –10,000 bytes. It can be said that there might be some similarities between the results of Figures 12 and 13 in those parts.

C. Result Applied to TCP Veno

TCP Veno (Vegas and ReNO) uses the *BufferSize* used by Vegas 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 follows.

$$\Delta cwnd = \begin{cases} 1 \text{ or } 0 (BufferSize > B) \\ 1 \ (BufferSize \leqq B) \end{cases} \quad (4)$$

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 $BufferSize \leqq B$.

We made the unidirectional packet trace from the trace in the previous experiment, and picked up a portion at the trace from 37.6 sec to 52.7 sec containing 23,261 data segments.

Figure 14 shows the relationship between *Δcwnd* and *cwnd* obtained in our previous paper. Figure 15 shows the relationship between *ΔsentData* and *sentData*. From the results, it can be said that *ΔsentData* are flat independent of *sendData*. Although the values of *ΔsentData* are not fit to zero and one segment, it can be said that there might be some similarities between the results of Figures 14 and 15.

V. CONCLUSIONS

This paper presented some studies on how to characterize the TCP congestion control algorithms from passively collected unidirectional packet traces. We applied our previous scheme, which compares *cwnd*, estimated from bidirectional packet traces, and its difference. Since we cannot estimate *cwnd* from unidirectional traces, the proposed scheme in this paper selects a short time period and treats the sent data size during this period as being proportional to *cwnd* at this time. Our scheme characterizes the TCP algorithm by use of the graph of*ΔsentData* and *sentData*.

We applied this scheme to TCP Reno and CUBIC TCP in detail and showed that our proposal seems to work. We also applied our scheme to HS TCP, TCP Vegas and TCP Veno.

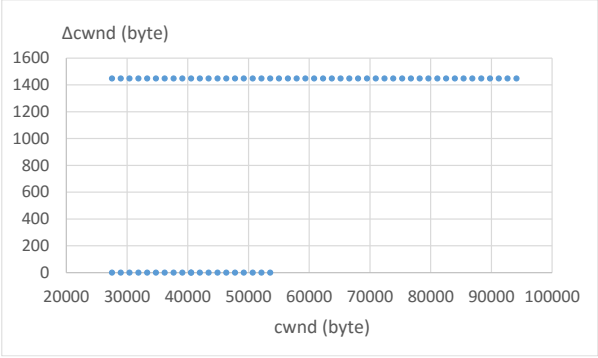

Figure 10. *Δcwnd* vs. *cwnd* for HS TCP [8].

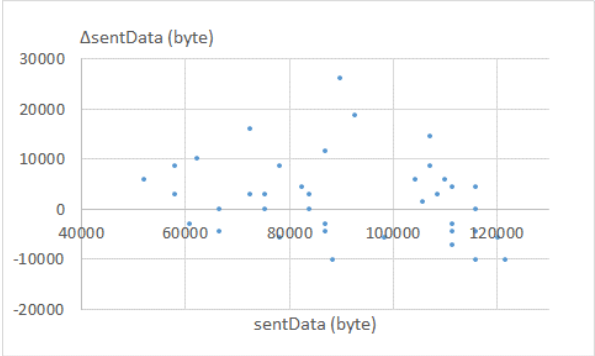

Figure 11. *ΔsentData* vs. *sentData* for HS TCP.

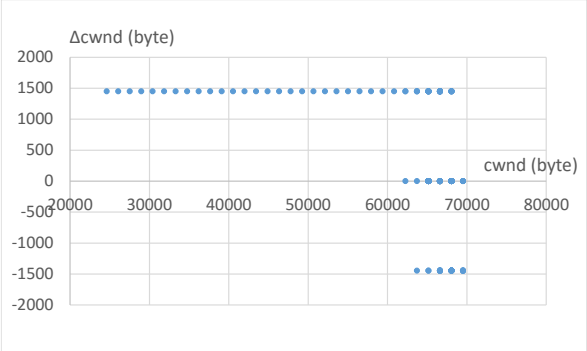

Figure 12. *Δcwnd* vs. *cwnd* for TCP Vegas [8].

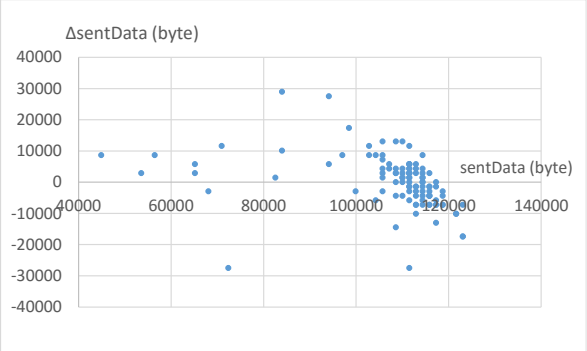

Figure 13. *ΔsentData* vs. *sentData* for TCP Vegas.

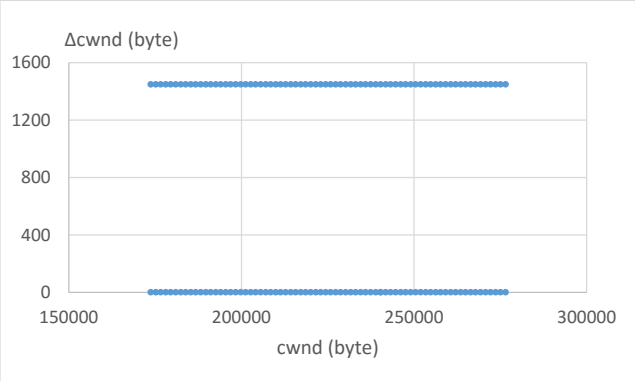

Figure 14. *Δcwnd* vs. *cwnd* for TCP Veno [8].

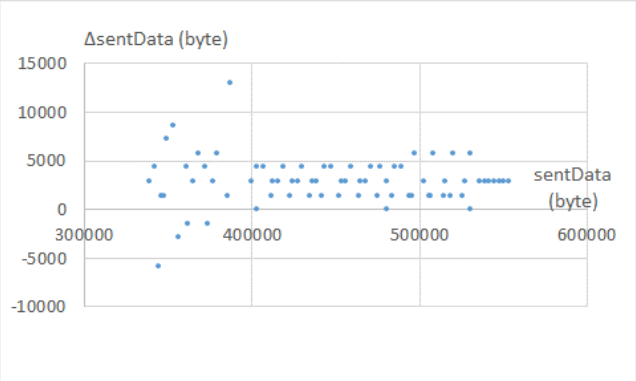

Figure 15. *ΔsentData* vs. *sentData* for TCP Veno.

The results were worse than those for Reno and CUBIC. For Vegas and Veno, the results in this paper have some similarity with the results in our previous paper, but there are some difficulties to distinguish Reno, Vegas and Veno. On the other hand, for HS TCP, our proposed scheme could not characterize it although our previous scheme could. It seems to characterize TCP Reno and CUBIC TCP algorithms well in our controlled in-laboratory setup. For the other TCP algorithms, the approach must be refined.

The points to be improved include the followings. First of all, the accuracy of our scheme needs to be increased. For this purpose, one important approach is to estimate a RTT correctly from unidirectional traces, because our scheme is sensitive for the fact that the short time period for calculating sent data is equal to RTT or integral multiples of RTT.

As describe above, the results in Figures 4, 13 and 15 have some similarities in the sense that the graphs have flat parts independent of *sentData*. So, it may be difficult to discriminate them. The next point is to invent a procedure to categorize the TCP algorithms into more general groups. , such as

- a flat type such as TCP Reno, HS TCP, TCP Westwood+ [16], TCP Vegas and Veno,
- a symmetric cubic root square type (CUBIC TCP),
- a monotonous increasing cubic root square type (Hamilton TCP [17]), and
- a random type (TCP Illinois [18]).

Thirdly, our previous scheme uses the estimation of multiplicative decrease parameter β is also used together with the estimation of window growth function. The scheme in this paper also needs to consider the multiplicative decrease parameter.

The last point is that the evaluation in this paper is done in a controlled laboratory setup. For the generalization of the results, the test has to be conducted under realistic network conditions

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. 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.

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

[6] 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.

[7] 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.

[8] T. Kato, A. Oda, C. Wu, and S. Ohzahata, "Comparing TCP Congestion Control Algorithms Based on Passively Collected Packet Traces," Proc. IARIA ICSNC 2015, Nov. 2015, pp.

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

[10] 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.

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

[12] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the Characteristics and Origins of Internet Flow Rates," Proc. ACM SIGCOMM'02, Aug. 2002, pp. 309-322.

[13] K. Lan and J. Heidemann, "Measurement Study of Correlations of Internet Flow Characteristics," Computer Networks, vol. 50, iss. 1, Jan. 2006, pp. 46-62.

[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] 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.

[17] 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.

[18] 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.

 ISBN: 978-1-61208-475-6