IADIS International Journal on Computer Science and Information Systems
Vol. 13, No. 2, pp. 16-29
ISSN: 1646-3692

PERFORMANCE EVALUATION OF TCP SPURIOUS TIMEOUT DETECTION METHODS UNDER DELAY SPIKE AND PACKET LOSS EMULATING LTE HANDOVER

Toshihiko Kato, Masahito Moriyama, Ryo Yamamoto, and Satoshi Ohzahata
University of Electro-Communications, 1-5-1, Chofugaoka, Chofu, Tokyo 182-8585, Japan

ABSTRACT

This paper describes the performance evaluation of the well-known spurious timeout detection methods implemented within TCP, Eifel, DSACK, and F-RTO, through experiments with the network emulator emulating handovers over LTE (Long Term Evolution) networks. Specifically, the emulator supports to insert the time-variant delay and packet loss in TCP streams. By taking account of the lossless handover in LTE, this paper shows the results for the cases only the delay spike is inserted, and both delay spike and packet loss are inserted. In the former case, the three methods show the similar performance, but in the latter case, the performance of Eifel is worse than the others. This paper also shows the results when two methods are used together for the delay spike and packet loss case, and indicates that the performance is not improved even if multiple spurious timeout detection methods are implemented.

KEYWORDS

TCP, Spurious timeout, Eifel, DSACK, F-RTO, LTE

1. INTRODUCTION

Modern wireless and mobile networks, such as IEEE 802.11 WLANs (Wireless LANs) (IEEE, 2016) and the 4G LTE (Long Term Evolution) wide area mobile network (Sesia et al., 2011), establish high speed and high quality data communication. Although previous wireless networks suffering from packet losses invoked by bit errors, modern wireless networks provide rich capabilities recovering from packet losses in their MAC (media access control) protocols. Moreover, the LTE protocols provide lossless handover functions that avoid packet losses even during handover changing the base stations called eNodeBs. So, it is possible that upper layer protocol modules over LTE networks do not experience packet losses during not only communication via one eNodeB but also handover between two eNodeBs.

Instead, during a handover, upper layer protocols over LTE, such as TCP and UDP, encounter sudden increase of delay, i.e., *delay spike*, in the order of seconds. Although it is highly possible that there are no packet losses during this large delay, TCP will retransmit a packet by timeout caused by this delay, and the timeout is not effective, i.e., *spurious*. The spurious timeout invokes unnecessary decrease of congestion window (*cwnd*) and reduces the TCP throughput. In order to detect the spurious timeout and recover unnecessary decrease of cwnd, several methods are proposed and implemented. Among them, the Eifel algorithm (Ludwig and Katz, 2000), (Ludwig and Reyer, 2003), the DSACK based algorithm (Blanton and Allman, 2004), and F-RTO (Sarolahti et al., 2003), (Sarolahti et al., 2009) are standardized as RFCs (requests for comments) by IETF (Internet Engineering Task Force) and implemented in several operating systems. So far, several papers discussed the performance of the spurious timeout detection methods, but many of them are using simulation, and the detailed performance evaluation is not reported over the LTE network.

In this paper, we show the results of performance evaluation through experiments using Linux PCs and the network emulator that provides time-variant delay and packet losses. We compare the Eifel algorithm, DSACK based algorithm, and F-RTO under the conditions that there is only delay spike, which supposes the LTE lossless handover, and that there are delay spike and some packet losses, which supposes some packet losses during handover. The rest of paper consists of the following section. Section 2 shows related work on the TCP spurious timeout detection methods and TCP performance evaluation. Section 3 describes the experimental conditions including the network emulator we implemented. Section 4 gives the results of performance evaluation, and section 5 concludes this paper.

2. RELATED WORK

2.1 Survey on Spurious Timeout Detection Methods

Table 1 shows examples of spurious timeout detection methods proposed so far. These methods are summarised as follows.

The Eifel algorithms relies on the TCP timestamp option. If retransmission timer expires, a TCP sender retransmits the oldest unacknowledged segment and remembers the timestamp of this segment. When an ACK segment is returned after the timeout, it compares the timestamp echo reply value in the ACK segment with the remembered timestamp value. If the echo reply is earlier than the timestamp value, then the sender recognises that the timeout was spurious. The DSACK based algorithm modifies the TCP SACK (selective acknowledgment) option's rule so as to allow the first SACK components to indicate the duplicate data segments. When a TCP sender receives an ACK segment, it compares the ACK number of the TCP header and the first component of SACK option. If the ACK number is larger, it means that the data segments corresponding to the first SACK component is received duplicated. In this case, the retransmission is considered as spurious. F-RTO does not rely on any TCP options. When a retransmission timer expires, a TCP sender enters the slow start phase and retransmits one data segment. When an ACK segment is returned for the retransmission, the sender sends two new, i.e., not previously transmitted, data segments. If the ACK segment received following these data segments responds to the new data, that is, if it is a new ACK, then the sender recognizes

that the timeout was spurious. Those three methods are currently standardized as RFCs and implemented widely. In this paper, we focus on them.

The STODER algorithm (Tan et al., 2005) exploits TCP repacketization to detect spurious timeouts. If a TCP sender detect the retransmission timer expiry, it retransmits a data segment which is *k*-bytes smaller than the original segment. It detects the spurious timeout, if the ACK segment after the timeout has the ACK number more than the last byte retransmitted. In the E-RTO algorithm (Lee and Kwon, 2006), a TCP sender transmits a new data segment on retransmission timeout, instead of the oldest unacknowledged segment. If the ACK segment for the new data segment acknowledges all of the outstanding data, then the timeout is considered as spurious. It also uses the SACK option in order to handle packet losses during the delay spike. If the TCP sender receives a duplicate ACK with SACK option, then it aggressively retransmits unSACKed segments. The ER-SRTO algorithm (Cho et al., 2008) also intends to detect packet losses during delay spike and to conduct efficient recovery. As for the spurious timeout detection, it uses the same approach with the Eifel algorithm. After that, if duplicate ACKs are received, it tries to retransmit outstanding data segments aggressively. The retransmission of the E-RTO and ER-SRTO algorithms might be too aggressive compared with the normal TCP fast retransmit. The ECN (Explicit Congestion Notification) nonce based algorithm (Welzl, 2008) uses the ECN nonce code point in the IP datagram header. The ECN capable sender sets the ECN nonce code point to 1 for original data segments, a value called ECN(1), and 0 (ECN(0)) for retransmitted data segments. ACK segments include the corresponding Nonce Sum field in TCP header, which allows the TCP sender to discriminate whether the retransmission was spurious or not. The SnR (Sprit-and-Retransmit) algorithm (Wen and Yeung, 2010) is similar with the STODER algorithm and its key idea is to split the retransmitted data segment into two smaller ones. A TCP sender detects the spuriousness when it receives an ACK segment with larger ACK number than the sequence number of the segment retransmitted at first.

The methods so far use an approach that a TCP sender detects the spuriousness by an ACK segment with some information, and that it resets the congestion control event if the timeout was spurious. In contrast, there are two proposals that increase delay intentionally when handover occurs (Klein et al., 2004), (Ahn et al., 2012). They focus mainly on the download data transfer to mobile nodes. Mobile nodes check the delay increase or the invocation of handover procedure, and inject delays to returning ACK segments. WiTracer (Hu et al., 2013) is a relatively new proposal that combines the congestion identification by use of RTT threshold, in the sender side, and the opportunistic recovery in the sender and receiver sides. The congestion identification relies on the timestamp option to detect RTT for every segment. As for the opportunistic retransmission, a sender invokes retransmission for a single duplicate ACK, and a receiver allows the spurious data reception for every other data or after a specific time period.

Table 1. Comparison of existing spurious timeout detection methods

name	option	RFC	detection time	modification
Eifel	Timestamp	RFC 3522	< RTT	Tx/Rx
DSACK based	SACK	RFC 3708	> RTT	Tx/Rx
F-RTO	N/A	RFC 5682	> RTT	Tx
STODER	N/A	N/A	< RTT	Tx
E-RTO	SACK	N/A	< RTT	Tx/Rx
ER-SRTO	Timestamp	N/A	< RTT	Tx/Rx
ECN based	IP ECN nonce bit	N/A	< RTT	Tx/Rx
SnR	N/A	N/A	< RTT	Tx
Delay Injection	Mobile protocols	N/A	N/A	Rx
WiTracer	Timestamp	N/A	< RTT	Tx/Rx

2.2 Studies on Performance Evaluation of TCP Over Wireless Networks

There are some studies on the performance evaluation of TCP over wireless networks. (Park and Chung, 2010) is an example of simulation-based study on TCP behavior over wireless multi-hop networks. It focuses on spurious fast retransmit recovery as well as spurious timeout retransmission, and indicates that spurious fast retransmit occurs more frequently when the number of hops is small.

On the other hand, other studies report the results of experiments using actual mobile nodes and actual wireless networks. (Kohlwes et al., 2005) shows the performance evaluation over the UMTS (Universal Mobile Telecommunications System) 3G network. It reports that RTT is fairly stable, and that spurious retransmissions were extremely rare. It concludes that no performance benefit was observed for the Eifel and F-RTO algorithms. (Li et al., 2015) and (Li et al., 2017) discuss TCP behaviors in the high speed (> 200 Km/h) mobility cases, in HSPA+ (high speed packet access plus) and 3G/4G, respectively. They show that a number of packet losses and connection shutdowns, but do not discuss about the effects of spurious timeout detection methods. In contrast, we use an actual Linux PC as a terminal and introduce a network emulator providing time-variant delay and packet losses, which are derived from the performance evaluation of a real TCP communication over an LTE network in Japan. We discuss the behaviour of spurious detection methods actually introduced in the Linux operating system.

3. EXPERIMENTAL CONDITIONS

3.1 Network Emulator for LTE Handover

As a network emulator that provides delay and packet losses, NetEm (Jurgelionis et al., 2011) and Dummynet (Carbone and Rizzo, 2010) are adopted widely. They are used in the middle of communicating nodes and can insert delay and packet losses, according to the specified parameter values and distributions. However, these values and distributions are static throughout a measurement run.

Figure 1 shows an example of delay performance measured in a Japanese LTE network, when a mobile node moving by a train in Tokyo area communicates with a server located in our lab. The horizontal axis is the time elapsed and the vertical axis RTT. This result is measured by PING where a mobile node sends ICMP Echo request packets with 500 byte data in the interval of 0.01 sec. The mobile node and the server synchronized the system clock by NTP (Network Time Protocol). The red line shows the one way delay from the mobile node to the server, and the green line indicate the one way delay in the reverse direction. This result indicates that the delay was normally small but there were several delay spikes during the measurement run. Although there were several packet losses, the timing was random, some were in the periods with small delay and others were during the delay spikes.

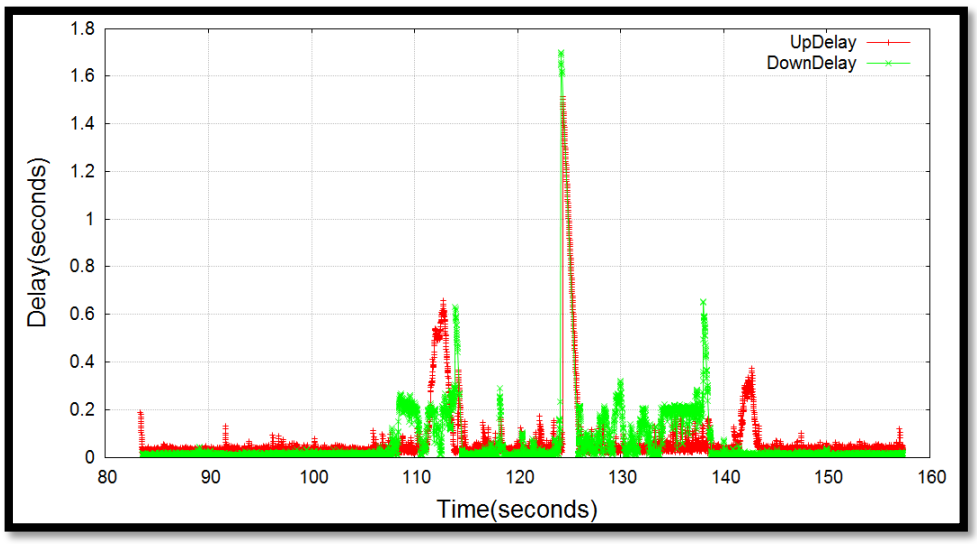


Figure 1. An example of delay measurement in an LTE network using a node moving around on a train in Tokyo area

In order to emulate the LTE handover as shown in Figure 1, we have developed a network emulator with the following functions.

- The network emulator supports the time-variant delay and packet loss rate. The delay and packet loss rate are given by a table as shown in Figure 2(a). It specifies the elapsed time (*pTime*) from the time handling the first packet by the emulator. The delay (*pDelay*) is the time period in which a packet arriving at the emulator at time *pTime* is queued in the

emulator. Similarly, the loss rate (*pLoss*) is the packet loss rate which a packet arriving at time *pTime* suffers from. For the timing not specified in the table, the delay and loss rate are interpolated linearly as shown in Figure 2(b). It should be noted that the delay and loss rate can be changed sharply by specifying different values for the same elapsed time.

- When a packet arrives at the emulator at time *t*, where *t* is between *pTime*[*i*] and *pTime*[*i*+1], it is assigned *delay* as given in Eq. (1). At the outgoing port in the emulator, the packet is queued for the period of *delay*.

$$delay\ at\ t = \frac{pDelay[i+1] \times (t - pTime[i]) + pDelay[i] \times (pTime[i+1] - t)}{pTime[i+1] - pTime[i]} \quad (1)$$

  In the case of Figure 2, packets arriving at the emulator during 10,000 msec to 11,460 msec are transmitted in a very short period during 15,000 msec and 15,010 msec.

- As for the packet loss rate, a packet arriving at time *t*, which is similar with above, is assigned the packet loss rate given by Eq. (2). When the packet is being transmitted from the emulator, it is discarded with this possibility.

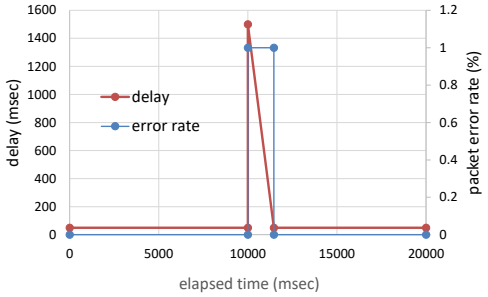
$$loss\ at\ t = \frac{pLoss[i+1] \times (t - pTime[i]) + pLoss[i] \times (pTime[i+1] - t)}{pTime[i+1] - pTime[i]} \quad (2)$$

  This mechanism is implemented using the Linux kernel function `random32()`, which generates a random number in the uniform distribution. The modulus of this random number with 10,000 is calculated (`random32()%10000`). If *loss* for this packet is smaller than the modulus, this packet is discarded at the outgoing port.

- We implemented the network emulator by modifying the source code of NetEm.

elapsed time: *pTime* (msec)	delay: *pDelay* (msec)	loss rate: *pLoss* (1/100 %)
0	50	0
10000	50	0
10000	1500	100
11460	50	100
11460	50	0
20000	50	0

(a) parameter setting for emulator



(b) delay and packet loss rate provided by emulator

Figure 2. Delay and packet loss rate added at emulator

3.2 Experiment Configurations and Parameters

Figure 3 shows the network configuration we used in the experiment. Two PCs are connected by Gigabit Ethernet via the network emulator described above. In the sender side, the packet transmission speed is limited to 10 Mbps with the traffic control (`tc`) command supported by the Linux operating system. In this experiment, the delay and packet loss are inserted only in

the direction from the sender to the receiver. In order to analyze the results in detail, the transmitted packets are captured at the output port of the sender, and the cwnd value is monitored in the sender by use of the tcpprobe function in Linux (Linux Foundation, 2016). As for the delay and packet loss rate used in the emulator, we used the setting shown in Figure 2. The congestion control algorithm used in the sender is TCP Reno.

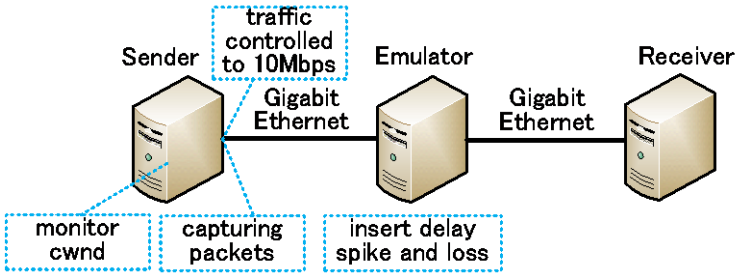


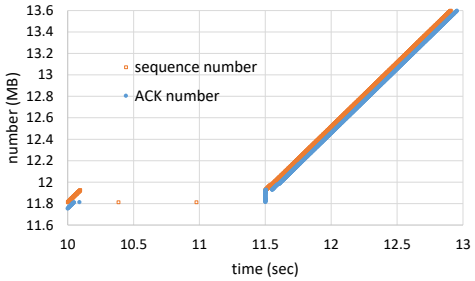
Figure 3. Network configuration of experiment

4. RESULTS OF PERFORMANCE EVALUATIONS

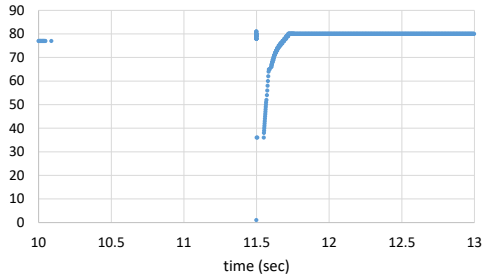
4.1 Experiment with Delay Spike Only

Figure 4 shows the results of performance evaluations when only delay spike is added at elapsed time 10 sec. The figure gives the time variation of the sequence/ACK numbers and cwnd for Eifel, DSACK, and F-RTO. The horizontal axis of graphs is the elapsed time, zooming up the duration between 10 sec. and 13 sec. The vertical axis is the sequence and ACK numbers counted in Megabyte and cwnd counted in packets. From the graphs indicating the cwnd values, there are some differences among three methods. F-RTO works best among the three. It can detect the spurious timeout and return to the previous cwnd value very quickly. In the case of Eifel, the spurious timeout can be detected quickly and cwnd is recovered to the value before the timeout, but after that, cwnd is halved and then increases to the previous value. Among three methods, DSACK is poor. The cwnd is reduced to one packet and then it goes up following the slow start and congestion avoidance procedures. This is an ordinal step after timeout, and so it can be said that no steps for spurious timeout detection are working.

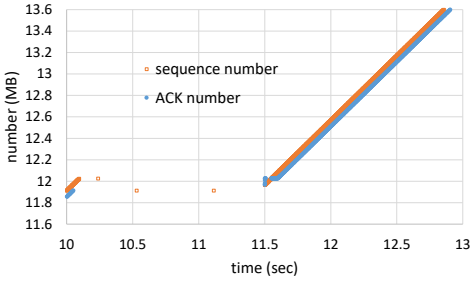
However, since RTT after the delay spike is small in this experiment, those differences in the cwnd behaviors are not reflected to the behaviors of sequence/ACK numbers. The increase situations of cwnd shown in Figure 4 (a), (c), and (e) are similar for the three methods. So, it can be said that Eifel, DSACK, and F-RTO work well in the LTE lossless handover.



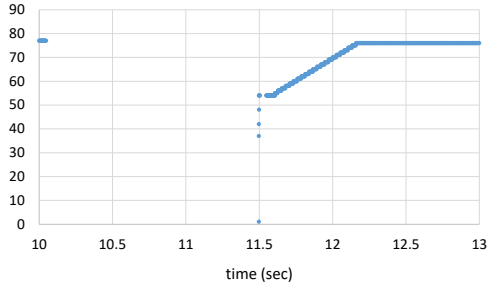
(a) sequence number and ACK number for Eifel



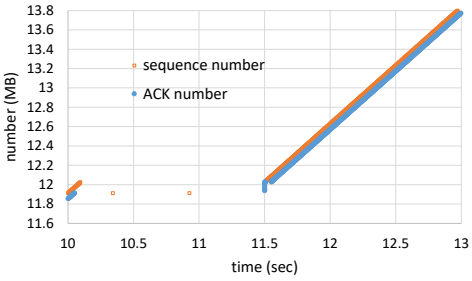
(b) cwnd (in packets) for Eifel



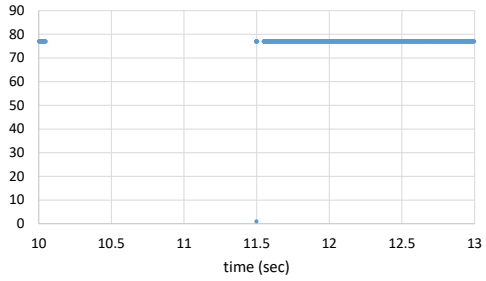
(c) sequence number and ACK number for DSACK



(d) cwnd (in packets) for DSACK



(e) sequence number and ACK number for F-RTO



(f) cwnd (in packets) for F-RTO

Figure 4. Sequence/ACK numbers and cwnd vs. time with delay spike only

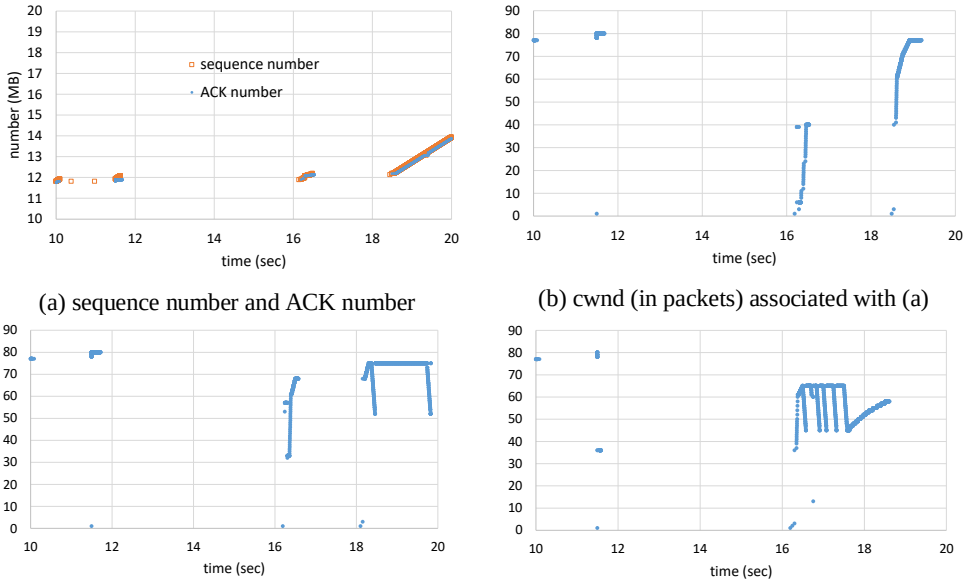
4.2 Experiment with Delay Spike and Packet Loss

Next is the case that packet losses are associated with the handover. Figures 5 through 7 shows the time variation of the sequence/ACK numbers and cwnd for Eifel, DSACK, and F-RTO, respectively. The specification of graphs are similar with Figure 4. In this case, we focus on the elapsed time between 10 sec. and 20 sec. because there are packet losses in this time frame. Since packet losses are provided by stochastic means in our emulator, we performed three measurement runs for each method. The figures show a sequence/ACK numbers vs. time result for one of the measurement runs and the cwnd vs. time results for three measurement runs.

As shown in Figure 5, Eifel provides the worst performance among the three methods. Figure 5(a) shows that, after the delay spike between 10 sec. and 11.5 sec., there is a long gap between 12 sec. and 16 sec. This is a time gap caused by a timeout retransmission. After that, there is

another timeout retransmission gap between 16.5 sec. and 18.5 sec. Figure 5(b) shows the cwnd vs. time result associated with Figure 5(a). At time 12 sec., cwnd is 80 packet, and it goes to one packet at time 16 sec. This shows the retransmission timeout decreased cwnd, and cwnd increases according to the slow start algorithm. When cwnd reaches 40 packet, there is another timeout retransmission, and cwnd goes to one packet. But in this case, cwnd returns to the value before the second retransmission. This means that Eifel works well and cwnd is recovered quickly. Figures 5(c) and 5(d) show the cwnd vs. time results in the other two measurement runs. The result in Figure 5(c) is similar with that in Figure 5(b). There is a long timeout gap after time 12 sec. and cwnd goes to one packet. There is another time gap after 16 sec. and after that, cwnd goes to one packet and returns to the cwnd value before the second retransmission quickly. The result shown in Figure 5(d) is a little different from the other two cases. There is only one timeout retransmission between 12 sec. and 16 sec., and after that, cwnd keeps around 65 packets with some decreases. Summarizing the above, when there are some packet losses with a delay spike, Eifel generates a long time gap resulting from a timeout retransmission and this degrades the performance largely.

Figure 6 shows the results when DSACK is used as a spurious timeout detection method. Figure 6(a) shows that, after the delay spike between 10 sec. and 11.5 sec., the sequence number increases steadily, but slowly compared with the case without packet losses. Figure 6(b) shows the cwnd vs. time result associated with Figure 6(a). Although there are several drops in a cwnd time variation, there are no gaps such as in Figures 5(b), 5(c) and 5(d) in the Eifel case. In this sense, the performance of DSACK is better than Eifel.



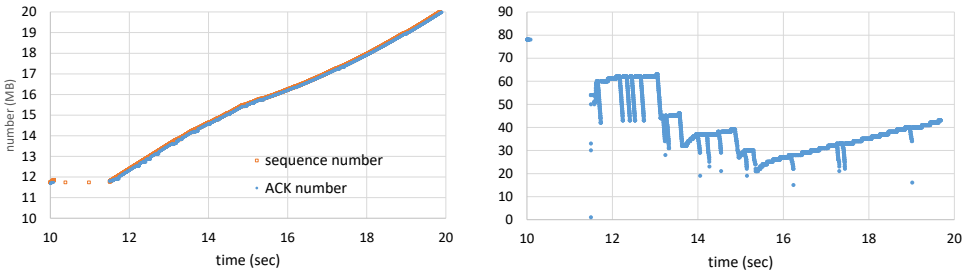
(a) sequence number and ACK number

(b) cwnd (in packets) associated with (a)

(c) cwnd (in packets) (another case)

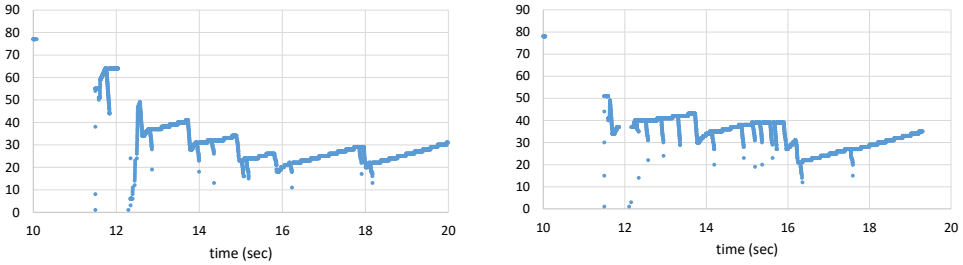
(d) cwnd (in packets) (yet another case)

Figure 5. Sequence/ACK numbers and cwnd vs. time with delay spike and packet loss for Eifel



(a) sequence number and ACK number

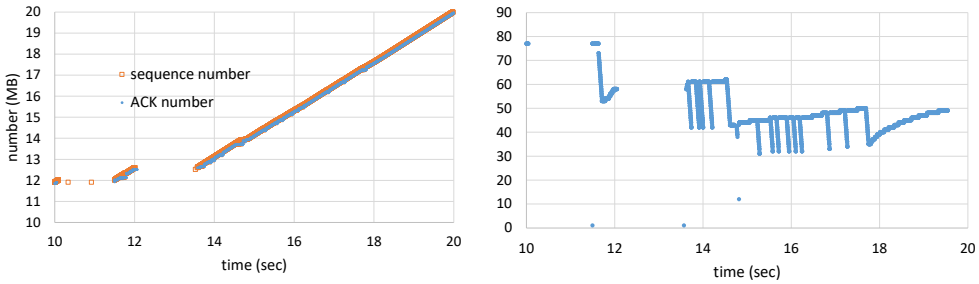
(b) cwnd (in packets) associated with (a)



(c) cwnd (in packets) (another case)

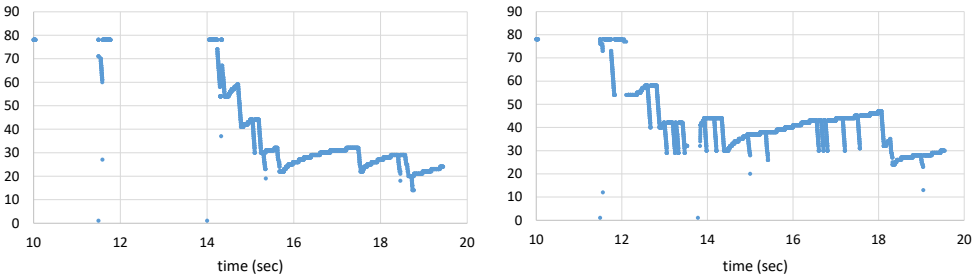
(d) cwnd (in packets) (yet another case)

Figure 6. Sequence/ACK numbers and cwnd vs. time with delay spike and packet loss for D-SACK



(a) sequence number and ACK number

(b) cwnd (in packets) associated with (a)



(c) cwnd (in packets) (another case)

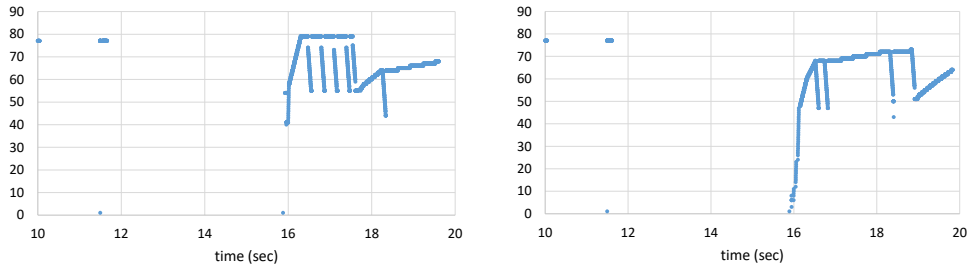
(d) cwnd (in packets) (yet another case)

Figure 7. Sequence/ACK numbers and cwnd vs. time with delay spike and packet loss for F-RTO

Figure 7 shows the results when F-RTO is used. Figure 7(a) shows that there is a time gap whose length is 1.5 sec. after the delay spike. This length of time gap in this case is smaller than the case in Eifel, and this is because, in the case of Eifel, no data segments are transferred after the delay spike, but some data segments are received after the delay spike in the case of DSACK. The retransmission timeout duration is increased according to the exponential backoff procedure in Eifel. Figure 7(b) shows the cwnd vs. time result associated with Figure 7(a). Although there is a gap in this graph and cwnd goes to one packet after the gap, cwnd recovers quickly to the value before the gap. It is considered that this is an effect of DSACK. Figures 7(c) and 7(d) show the result of cwnd vs. time in the other two measurement runs. The result in Figure 7(c) has one timeout gap after the delay spike and the cwnd value after the gap is smaller than the case in Figure 7(b). In the result given in Figure 7(d), there are no timeout gaps after the delay spike, but cwnd decreases multiple times due to the fast retransmit. The performance in this case is similar with that in DSACK, which is better than the Eifel case.

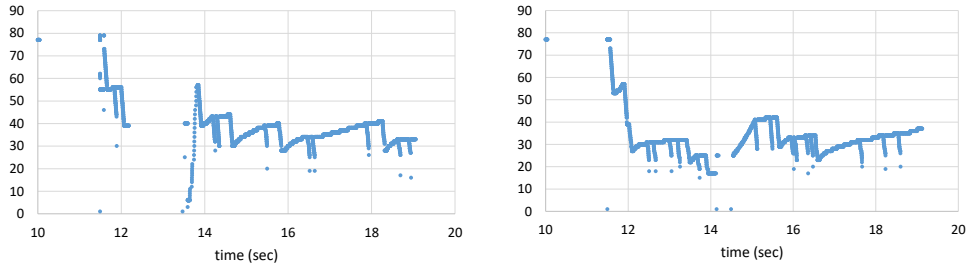
It is possible to introduce multiple methods to detect spurious timeout, and so we tried to use two methods together. Figures 8, 9, and 10 show the cwnd vs. time results when Eifel and DSACK, Eifel and F-RTO, and DSACK and F-RTO are used together, respectively. Each of the figures shows the results of two measurement runs. Figure 8 shows that, when Eifel is used together with DSACK, there is a long time gap, which is invoked by a repeated timeout retransmission of the first data segment lost during the delay spike. This situation is similar with the case where Eifel is used alone. This means that the introduction of DSACK is not effective for the use of Eifel when some packet losses occur during a delay spike emulating an LTE handover.

Figure 9 shows that, when F-RTO is used with Eifel, the retransmission timeout based long time gap may disappear after the delay spike accompanying packet losses. That is, F-RTO can recover the drawback of Eifel due to the packet losses. But the value of cwnd is not large and similar with the value in the case that F-RTO is used by itself.



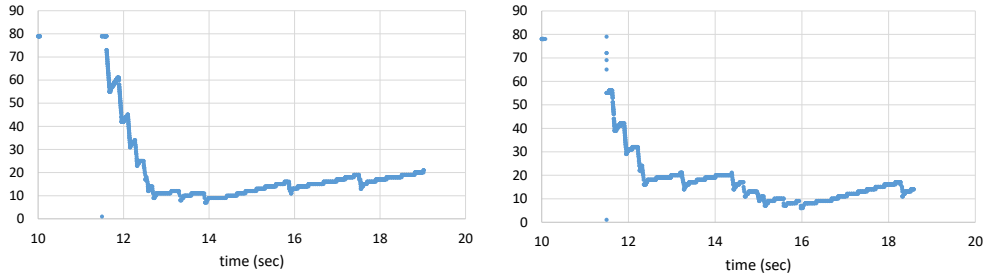
(a) cwnd (in packets) (b) cwnd (in packets) (another case)

Figure 8. cwnd vs. time with delay spike and packet loss for Eifel and DSACK



(a) cwnd (in packets) (b) cwnd (in packets) (another case)

Figure 9. cwnd vs. time with delay spike and packet loss for Eifel and F-RTO



(a) cwnd (in packets) (b) cwnd (in packets) (another case)

Figure 10. cwnd vs. time with delay spike and packet loss for DSACK and F-RTO

When DSACK and F-RTO are used together, as shown in Figure 10, there are no time gaps after the delay spike with packet losses, but the cwnd decreases to lower values compared with the cases either DSACK or F-RTO is used by itself. All of these results show that the combination of two spurious timeout detection methods does not improve the performance after a delay spike accompanying packet losses.

Figure 11 shows the transferred data size (MB) during a measurement run whose duration is 20 sec. The value is an average among three runs. This result says that DSACK and F-RTO provide the best performance among the three methods. It also says that the combination of two spurious timeout detection methods does not increase the throughput.

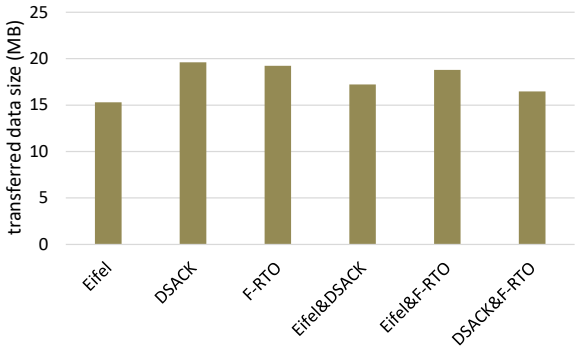


Figure 11. Transferred data size during measurement run

5. CONCLUSIONS

In this paper, we presented the results of performance evaluation for TCP spurious timeout detection methods supposing LTE handovers. We adopted a way to use real communicating nodes with introducing a network emulator that provides time-variant delay and packet loss, a delay spike and packet losses during the delay, to imitate LTE handovers. We implemented this network emulator over NetEm, a widely used freeware network emulator.

We conducted experimental measurements for Eifel, DSACK, and F-RTO, wide spread methods standardized by IETF, in the conditions with delay spike only and with delay spike and packet losses. When only the delay spike is inserted, the three methods work well similarly. When the packet losses are added with delay, the performance of Eifel is poorer than DSACK and F-RTO. We also conducted the performance evaluation by combining two methods together for the case that delay spike accompanies packet losses, and the results indicated that the combination does not increase the throughput.

ACKNOWLEDGEMENT

The authors thank Dr. M. Nomoto, Mr. K. Ohata, and Mr. R. Fujiyama for their efforts in the early stage of this project, especially the implementation of the network emulator described in Section 3 and the performance evaluation of delay characteristics of an LTE network given in Figure 1.

REFERENCES

Ahn, W., Gwak, Y., and Kim, Y., 2012. A Low-Complexity Delay Injection Algorithm for Improving TCP Performance During LTE Intra Handover. *International Conference on Information Network 2012*, pp. 177-181.

Blanton, E. and Allman, M., 2004. *Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions.* IETF RFC 3708.

Carbone, M. and Rizzo, L., 2010. Dummynet Revisited. *ACM SIGCOMM Comput. Commun. Rev*., vol. 40, issue 2, pp. 12-20.

Cho, I., Han, J., and Lee, J., 2008. Enhanced Response Algorithm for Spurious TCP Timeout (ER-SRTO). *International Conference on Information Networking, 2008 (ICOIN 2008)*, pp. 1-5.

Hu, C., Yang, X., Fan, M., and Zhao, P., 2013. WiTracer: A Novel Solution to Improve TCP Performance over Wireless Network. *2013 9th International Wireless Communications and Mobile Conputing Conference (IWCMC)*, pp. 450-455.

IEEE Std 802.11-2016, 2016. *IEEE Standard for Information technology – Part11: Wireless LAN medium Access Control (MAC) and Physical Layer (PHY) Specifications*.

Jurgelionis, A., Laulajainen, J., Hirvonen, M., and Wang, A., 2011. An Empirical Study of NetEm Network Emulation Functionalities. *20th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1-6.

Klein, T., Leung, K., Parkinson, R., and Samuel, L., 2004. Avoiding Spurious TCP Timeouts in Wireless Networks by Delay Injection. *IEEE Global Telecommunications Conference, 2004 (GLOBECOM '04)*, pp. 2754-2759.

Kohlwes, M., Riihijarvi, J., and Mahonen, P., 2005. Measurements of TCP Performance over UMTS Networks in Near-Ideal Conditions. *2005 IEEE 61st Vehicular Technology Conference*, vol. 4, pp. 2235-2239.

Lee, M. and Kwon, O., 2006. E-RTO: An Enhanced TCP Retransmission Timeout Algorithm using SACK option. *2006 IEEE Wireless Communications and Networking Conference (WCNC 2006)*, pp. 74-79.

Li, L., et al., 2015. A Measurement Study on TCP Behaviors in HSPA+ Networks on High-Speed Rails. *2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 2731-2739.

Li, L., et al., 2017. A Longitudinal Measurement Study of TCP Performance and Behavior in 3G/4G Networks over High Speed Rail. *IEEE/ACM Trans. on Networking*, vol. 25, no. 4, pp. 2195-2208.

Linux Foundation, 2016. *Linux Foundation Wiki, Trace: tcpprobe*. https://wiki.linuxfoundation.org/networking/tcpprobe.

Ludwig, R. and Katz, R., 2000. The Eifel algorithm: making TCP robust against spurious retransmissions. *ACM SIGCOMM Comput. Commun. Rev.*, vol. 30, issue 1, pp. 30-36.

Ludwig, R. and Reyer, M., 2003. *The Eifel Detection Algorithms for TCP*. IETF RFC 3522.

Park, M. and Chung, S., 2010. A Simulation-based Study on Spurious Timeouts and Fast Retransmits of TCP in Wireless Networks. *2010 Third International Joint Conference on Computational Science and Optimization*, pp. 273-277.

Sarolahti, P., Kojo, M., and Raatikainen, K., 2003. F-RTO: an enhanced recovery algorithm for TCP retransmission timeouts. *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, issue 2, pp. 51-63.

Sarolahti, P., Kojo, M., Yamamoto, K., and Hata, M., 2009. *Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP*. IETF RFC 5682.

Sesia, S., Toufik, I., and Baker, M., 2011. *LTE – The UMTS Long Term Evolution, From Theory to Practice, Second Edition.* A John Wiley & Sons, Ltd., Publication.

Tan, K., Zhang, Q., and Zhu, W., 2005. STODER: a robust and efficient algorithm for handling spurious retransmit timeouts in TCP. *IEEE Global Telecommunications Conference, 2005 (GLOBECOM '05)*, pp. 3692-3696.

Welzl, M., 2008. Using the ECN Nonce to detect Spurious Loss Events in TCP. *IEEE Global Telecommunications Conference, 2008 (GLOBECOM '08)*, pp. 1-6.

Wen, Z. and Yeung, K., 2010. On Detection Algorithms for Spurious Retransmissions in TCP. *2010 IEEE Wireless Communications and Networking Conference (WCNC 2010)*, pp. 1-6.