Resolving Bufferbloat in TCP Communication over IEEE 802.11n WLAN by Reducing MAC Retransmission Limit at Low Data Rate

Masataka Nomoto, Celimuge Wu, Satoshi Ohzahata, and Toshihiko Kato
University of Electro-Communications
Tokyo, Japan
e-mail: noch@net.is.uec.ac.jp, clmg@is.uec.ac.jp, ohzahata@is.uec.ac.jp, kato@is.uec.ac.jp

***Abstract*— IEEE 802.11n wireless local area networks (LANs) provide the data transmission rate of hundreds of Mbps. At the same time, they support multiple data rates and the dynamic rate switching functionality in order to cope with various radio conditions. However, a low data rate may cause a long delay in transmission control protocol (TCP) communications, which is called a bufferbloat problem. In this paper, we infer that one possible reason for the delay is the powerful retransmission capability supported by 802.11n, and propose a method which weakens this capability intentionally for TCP communications when the data rate is low. This paper evaluates the performance of our proposal, the native 802.11n, and CoDel, which is an active queue management approach coping with the bufferbloat problem. It shows that CoDel and our proposal improve the delay performance and that CoDel sometimes reduces the throughput under a high data rate condition.**

Keywords- Wireless LAN; IEEE 802.11n; TCP; Dymamic Rate Switching; Bufferbloat Problem; Block Acknowledgment.

I. INTRODUCTION

Recently, wireless LANs (WLANs) conforming to the IEEE 802.11n standard [1] are being used widely. This type of WLANs can provide a data rate of hundreds of Mbps. In order to realize high throughput, 802.11n has added new physical and media access control (MAC) technologies to the conventional IEEE 802.11. They include multiple-input and multiple-output (MIMO), the channel bonding, the frame aggregation, and the block acknowledgment (Block ACK).

On the other hand, IEEE 802.11n supports multiple data rates and the dynamic rate switching to use the optimal data rate between a terminal and an access point (AP). When a terminal is located close to an AP and the radio condition is good, the high data rate such as 300 Mbps can be used. But, when a terminal moves to the location far from an AP and the receiving radio signal strength becomes weak, the data rate gets lower, for example down to 6.5 Mbps.

In our previous paper [2], we gave a detailed analysis of the performance of TCP communication during which a terminal changes the distance from an AP. As a result, when the distance between the terminal and the AP is large (e.g., 10 m), the packet losses do not increase, but the round-trip time (RTT) increases largely, up to several seconds. This long delay is considered as a sort of bufferbloat problem, which is discussed widely in the networking community [3]-[5]. In order to solve the bufferbloat problem, the active queue management is considered to be effective and an approach named CoDel is proposed [6]. CoDel uses a packet-sojourn time in a queue as a control parameter, and drops a packet in the situation when packets stay too long in the queue.

Our previous paper [2] suggested a different approach from the active queue management. We inferred that one of the reasons for the large queuing delay is the powerful data retransmission function in 802.11n MAC level, which uses the frame aggregation and the Block ACK. So, we proposed that it would be possible to resolve the bufferbloat problem by intentionally weakening the capability of retransmission realized by Block Ack frames, only when the data rate is low in TCP communications. Specifically, we set the retransmission limit to 2 when the data rate is smaller than 80 Mbps, and use 10, which is the default value, when larger than 80 Mbps. Our previous paper showed that this scheme introduces MAC level frame losses and, as a result, reduces the RTT resulting from the shrunk congestion window size. However, this proposal is premature because it uses only two values for the retransmission limit. As for the performance evaluations, our previous work is also premature because it provides only a limited number of measurements.

In this paper, we propose a revised algorithm for reducing the delay in TCP communication over 802.11n WLAN. It defines intermediate values of the retransmission limit corresponding to the data rates between low and high ones, by use of linear interpolation in the semilog relation of data rate and retransmission limit. This paper also presents the detailed performance evaluation of our proposal. In the evaluation, a terminal is located in several positions with different distances from an AP, and the performance is measured for the proposal, CoDel, the native 802.11n, for TCP Reno and CUBIC TCP [7].

The rest of this paper is organized as follows. Section II explains the problem we focused on in this paper and the possible solutions proposed so far. Section III describes our proposed scheme for resolving bufferbloat problem for 802.11n WLAN, and Section IV gives the performance evaluation. In the end, Section V concludes this paper.

II. BUFFERBLOAT PROBLEM AND RELATED WORK

A. Bufferbloat problem in 802.11n WLAN

Table I gives the data rates supported by the terminal and the AP used in the experiment. In these data rates, an 802.11n data sender performs retransmission of corrupted frames. During this procedure, the data sender monitors the ratio of retransmissions and selects the lower data rate if the retransmission ratio becomes too large.

In this paper, we focus on the bufferbloat problem in the upload data transfer from a terminal to an AP. Consider the

 ISBN: 978-1-61208-546-3

TABLE I. AVAILABLE DATA RATE IN 802.11N WLAN.

6.5	13.5	27.0	40.5	54	81
108	162	216	243	270	300

Unit: Mbps

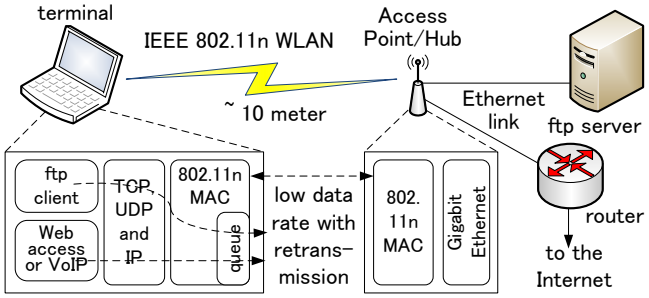


Figure 1. Outline of bufferbloat problem in 802.11n WLAN upload traffic.

situation depicted in Figure 1. A terminal located far from an AP is sending data to the ftp server. The terminal uses a low data rate, such as 6.5 Mbps and 13.5 Mbps. Our previous experiment gave some results that the retransmission at MAC level works well and there are few packet losses at the TCP and IP level [2]. Consequently, the TCP congestion window size grows up, and the data frames corresponding to this size are transmitted contiguously. However, the data rate is low and therefore the data frames are stored in the queue at the MAC level. This brings a large delay in the file transfer. If another application such as web access and voice over IP (VoIP) starts in this situation, the new communication also suffers from the large delay.

B. Related work

There are several approaches which can be applied to the problem described above.

The first one is the introduction of IEEE 802.11e [8]. It provides the priorities in the MAC level, i.e., Voice, Video, Best Effort, and Background, by introducing separate queues within a node and separate flows of data frames in a WLAN. In order to introduce separate flows, it discriminates values of arbitration interframe space and contention window boundaries. As for the bufferbloat problem, however, it cannot be always applied. If the second application in Figure 1 is TCP based, such as a web access, the ftp client and the second application are categorized in the same priority in 802.11e. So, the second application will suffer from the delay which the ftp generates.

The second approach is the active queue management. As described above, CoDel uses packet-sojourn time in the queue. Specifically, when any packet stays in the queue longer than a specific duration, called *target* in CoDel, during a predefined interval, called *interval* in CoDel, the last packet in the queue is dropped. As for the value of *target*, 5 msec is used in [6]. For the *interval*, 100 msec is used as the beginning of the procedure and, if a packet is dropped, the value is decreased in inverse proportion to the square root of the number of drops since the dropping state was entered. Some simulation results are shown in [6] over WiFi links whose data rate changes among 100Mbps, 50Mbps and 1Mbps, and tell that the per-packet queue delay in CoDel is smaller than that in random early discard (RED) [9] and Tail Drop.

The third approach is the adoption of TCP based on non-loss based congestion control. As described above, the grown congestion window size is the reason for queued data frames, and no loss situation allows the window size to grow. So, the introduction of non-loss based congestion control, such as TCP Vegas [10], might be effective. With the current values of congestion window size and RTT, TCP Vegas estimates the buffer size in the bottleneck node. A TCP sender increases the congestion window size when the bottleneck buffer size is small and decreases when the buffer size is large.

In contrast with those approaches, our scheme uses the retransmission limit adjustment. There are several studies focusing on this topic [11]-[13]. However, all of them focus on the relationship between the transmission delay and the retransmission limit. On the other hand, our scheme aims at causing a packet loss intentionally by changing the retransmission limit.

III. PROPOSAL

The basic idea of our scheme is that a MAC data sender tunes up the retransmission limit in response to the data rate used for data frame transmission. The lower data rate, the smaller retransmission limit. This adjustment is done only if the sending data frame contains a TCP segment by checking the protocol field in IP header. The followings give the points of our scheme.

A. Focusing on Block Ack based retransmission

As for the reception confirmation, IEEE 802.11n adopts an approach called High Throughput (HT)-immediate Block Ack [1]. A sender aggregates multiple data frames into one frame (aggregated MAC protocol data unit: A-MPDU) and sends it out. A receiver checks the correctness of individual received data frames, and returns a Block Ack frame. The Block Ack frame is sent out immediately after the receiver received the A-MPDU, and indicates individual data frames are received successfully or not in the Block Ack Bitmap field.

If the Block Ack Bitmap field indicates loss of some data frames, the sender side retransmits the lost frames (*the Block Ack based retransmission*). On the other hand, in the case when A-MPDU itself is corrupted or the returning Block Ack frame is lost, the A-MPDU is retransmitted again (*the timeout based retransmission*).

In general, the timeout based retransmission is controlled by a WLAN hardware chip and the Block Ack based retransmission is controlled by a WLAN device driver. They are managed independently. In the case of the WLAN device driver we use in this paper, the retransmission limit is 19 for the timeout based, and 10 for the Block Ack based retransmission.

Since our scheme is implemented in a WLAN device driver, we focus on the Block Ack based retransmission. Our scheme decreases the limit for the Block Ack based retransmission when the data rate becomes low.

 ISBN: 978-1-61208-546-3

B. *Determining retransmission limit for individual data rate*

The next point is what value is selected as the retransmission limit for an individual data rate. As described above, the maximum value of the Block Ack based retransmission is 10. On the other hand, our experiment described in [2] showed that 2 is appropriate as the retransmission limit for the data rate 6.5 Mbps and 13.5 Mbps. So, in this proposal, we focus on determining the in-between retransmission limit values.

We have decided to define the Block Ack based retransmission limit in the following way.

- For the data rate equal to and higher than 100 Mbps, the limit is 10.
- For the data rate equal to and lower than 10 Mbps, the limit is 2.
- As a first step, we introduce a linear relationship between the limit and the data rate between 10 Mbps and 100 Mbps over a semilog scale. This is depicted as a dashed line in Figure 2.
- Based on this result, we have selected stepwise values for the retransmission limit as shown by a solid line in the figure.

That is, the Block Ack based retransmission limit is

10 if $rate \geq 100$ Mbps,
8 if 50 Mbps $\leq rate < 100$ Mbps,
5 if 25 Mbps $\leq rate < 50$ Mbps, and
2 if $rate < 25$ Mbps.

It should be noted that this limit value selection is not based on a specific theory. However, the results given in Section IV show that our scheme works well using those limit values.

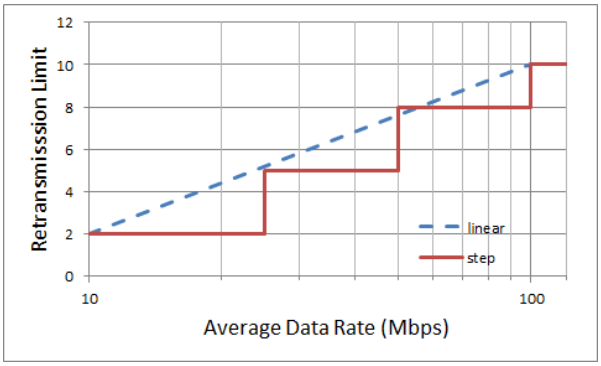


Figure 2. Retramission limit adopted by our scheme.

C. *Using moving average for data rate*

The last point is what data rate is to use for determining the retransmission limit. The data rate for a specific data frame is determined when the device driver handles the corresponding data transfer request. The data rate will be changed according to the physical layer status between the terminal and AP. So, we decided to introduce the exponential moving average with coefficient 0.25. That is, the *rate* described above is calculated at each of data transfer request by the following equation.

$$rate \leftarrow 0.75 \times rate + 0.25 \times actual\ data\ rate$$

The retransmission limit is determined using this rate and is applied when a data frame is retransmitted according to the Block Ack based retransmission.

IV. Performance Evaluation

A. *Experimental settings*

Figure 3 shows the network configuration of our experiment. A terminal and an AP use 5GHz band WLAN conforming to IEEE 802.11n. The AP and a server are connected via Gigabit Ethernet link through a bridge. The bridge is used to add a delay to emulate a communication via the Internet.

The experiment is performed in a two-storied Japanese style house built of wood. The server, the AP and the bridge are located in the 2nd floor. The terminal is located in various locations in the 1st and 2nd floors, and the stairs between them. The distance between the terminal and the AP is about 1.2 meter at the nearest position and about 10 meter at the far most position. At one position, the terminal is fixed and sends data to the server for 60 seconds. The data communication is done by use of iperf [14].

The specification of the terminal is given in Table II. The AP is commercially available and its model number is WZR-HP-AG300H manufactured by Buffalo Inc., Japan. This AP supports multi-rate up to 300 Mbps. In the experiment, we used all of the 12 levels of data rate given in Table I.

In the experiment, the performance of the proposed scheme, CoDel and the native 802.11n are evaluated. The detailed conditions of the experiment are as follows.

- The proposed scheme is implemented in the ath9k device driver [15].
- The CoDel used is that for Linux 3.5. We ported this version of CoDel to Linux 3.2.38. As the performance parameters in CoDel, we used default parameters, e.g., 5 msec as the target and 100 msec as the interval.

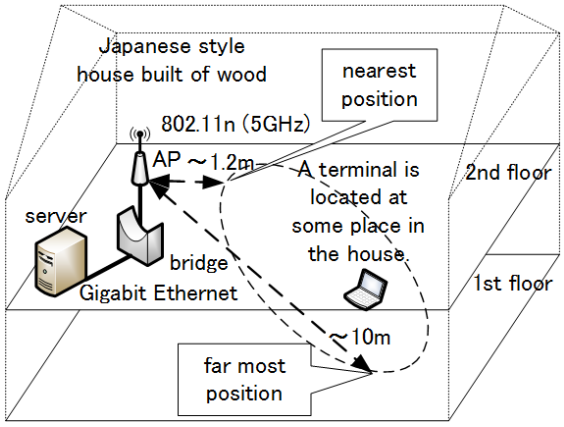


Figure 3. Experiment configuration.

TABLE II. SPECIFICATION OF TERMINAL.

Linux kernel	3.2.38 (self build)
Manufacturer/model	Lenovo ThinkPad X61
WLAN card	NEC Aterm WL300NC
WLAN device driver	ath9k

- As for TCP versions, we adopted TCP Reno, as a conventional scheme, and Cubic TCP, as the default in Linux.
- In the experiment, two cases without and with additional delay are evaluated. The delay is inserted by the bridge. In the case of additional delay, 100 msec round-trip delay (50 msec one way delay) is used. The insertion is done using netem in Linux [16].
- During a 60 sec. TCP communication, the following data are collected;
  - packet trace at the terminal, by use of tcpdump,
  - TCP connection information, such as the congestion window size (cwnd) at the terminal, by use of tcpprobe [17], and
  - WLAN transfer information, such as data rate, from WLAN device driver.

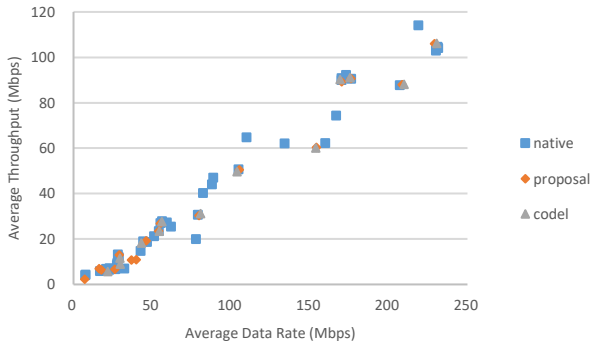
  From these data, the average of data rate, RTT, throughput, and cwnd for an individual TCP communication are calculated.
- As for the parameter which characterizes the position of the terminal, the distance between the terminal and AP is not appropriate. The reason is that the distance is only meaningful in our experimental environment. On the other hand, the data rate used in one position is rather stable. So, we use the average data rate during a TCP communication as the parameter which specifies the location of the terminal. The other measured values are mapped with the average data rate.

B. *Comparison among proposal, CoDel and native 802.11n*

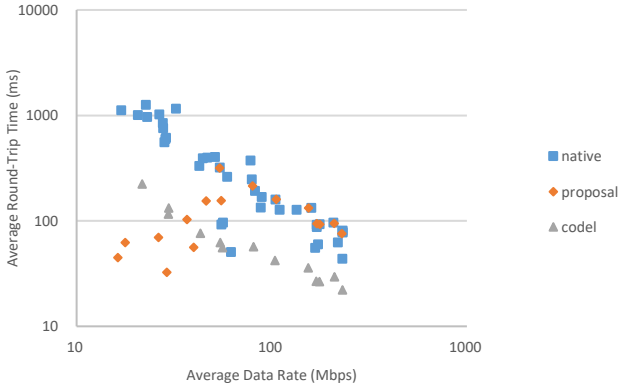
Figure 4 shows the results when Cubic TCP is used and no additional delay is inserted at the bridge. In this figure, (a), (b) and (c) show the average throughput, the average RTT, and the average cwnd versus the average data rate, respectively. An individual point in the figure shows a result of one evaluation for one 60 sec. TCP communication. From Figure 4 (a), it can be said that our proposal, CoDel, and the native 802.11n give a similar TCP throughput.

But, Figure 4 (b) indicates that the average RTT of the native 802.11n is large, about 1000 msec, when the average data rate is lower than 30 Mbps. The average RTT of CoDel is smaller than that of the native 802.11n for all values of the average data rate. The average RTT for the native 802.11n and CoDel maintains a linear relationship with the average data rate in the log-log scale. On the other hand, our proposal shows different features. In our proposal, the average RTT is similar with that of the native 802.11n while the average data rate is larger than 80 Mbps. For the average data rate smaller than 80 Mbps, however, the average RTT of our proposal becomes smaller than that of the native 802.11n, and even that of CoDel.

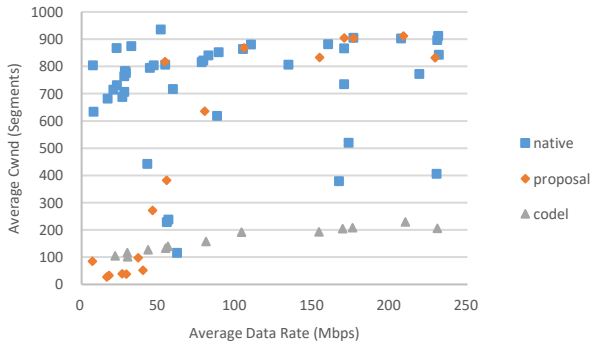
Figure 4 (c) shows the reason of those results for RTT. In the native 802.11n, the average cwnd is large, 700 to 900 segments, for all values of the average data rate. This large cwnd causes the queue to build up. In the case of CoDel, the average cwnd is small throughout all the range of the average data rate. This is caused by dropping packets against the built up queue. On the contrary, in our proposal, the average cwnd is similar with that of the native 802.11n while the average



(a) Average throughput versus average data rate



(b) Average RTT versus average data rate



(c) Average congestion window size versus average data rate

Figure 4. Results for Cubic TCP without any additional delay.

data rate is 100 Mbps or larger. When the average data rate becomes smaller than 100 Mbps, the average cwnd also becomes smaller, and in the range of below 40 Mbps, it is smaller than that of CoDel. It can be said that the proposed scheme to decrease the MAC level retransmission limit at the low data rate works well for a TCP communication.

When TCP Reno is used, the results were similar when no additional delay is inserted.

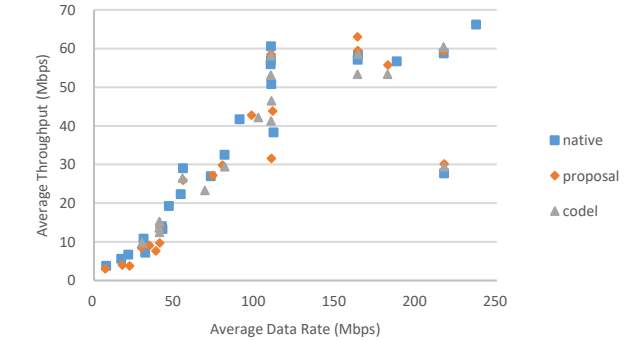
Figure 5 shows the results when Cubic TCP is used and 100 msec additional delay is inserted at the bridge. As for the average RTT, the native 802.11n has a large value and the CoDel is smaller than that of the native 802.11n, while the average data rate is smaller than 100 Mbps. On the other hand,

 ISBN: 978-1-61208-546-3

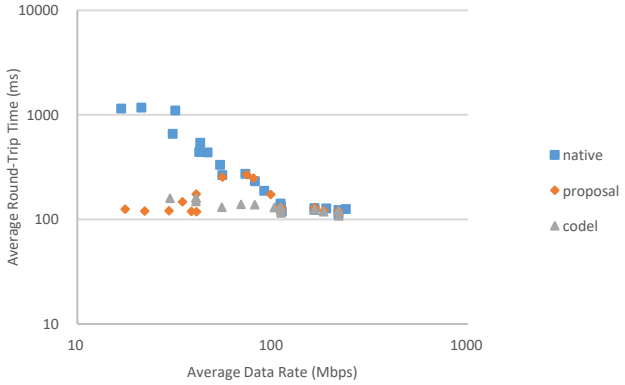
our proposal has similar average RTT values with the 802.11n while the average data rate is larger than 80 Mbps. For the average data rate smaller than 80 Mbps, however, the average RTT of our proposal becomes smaller than that of the native 802.11n, and even that of CoDel. This is similar with the case of Figure 4.

Figure 5 (c) gives a different result from Figure 4 (c). The average cwnd of CoDel in the case of additional delay is larger than the case without additional delay. The average cwnd of CoDel is similar with the native 802.11n and our proposal for 100 Mbps and larger average data rate. This brings the similar TCP throughput.

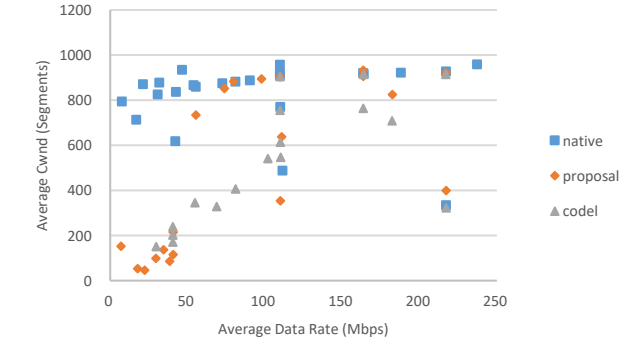
Figure 6 shows the results when TCP Reno is used and when 100 msec additional delay is inserted at the bridge. Figure 6 (b) shows that the average RTT is similar with that in Figure 5 (b). From Figure 6 (a), however, the average throughput of CoDel is lower than the other schemes in the range of the average data rate with larger than 100 Mbps. Figure 6 (c) shows that the average cwnd of CoDel is also smaller than those of our proposal and the native 802.11n for the average data rate larger than 100 Mbps. This is the reason for the low throughput. In order clarify the situation, Figure 7 shows the timeline of throughput and cwnd when the average data rate is 216 Mbps. Figure 7 (a) shows that the throughput of CoDel becomes low at time 15 sec. Figure 7 (b) indicates that, at this timing, a packet loss causes slow start and, after that, cwnd grows up only slowly. This result says that CoDel may drop packets unnecessarily and the TCP version with the moderate congestion increasing may suppress the throughput.



(a) Average throughput versus average data rate

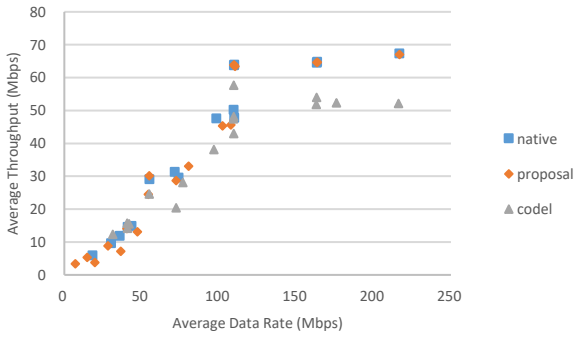


(b) Average RTT versus average data rate

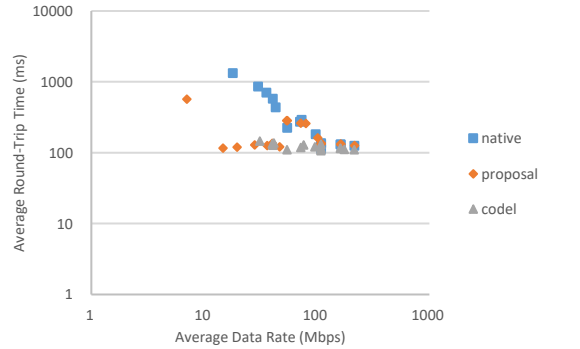


(c) Average congestion window size versus average data rate

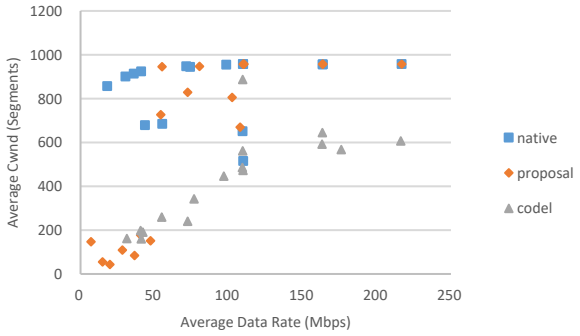
Figure 5. Results for Cubic TCP with 100 msec additional delay.



(a) Average throughput versus average data rate



(b) Average RTT versus average data rate

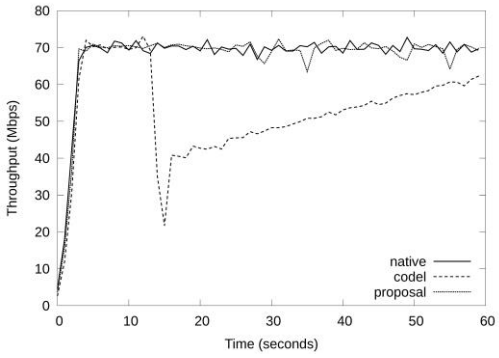


(c) Average congestion window size versus average data rate

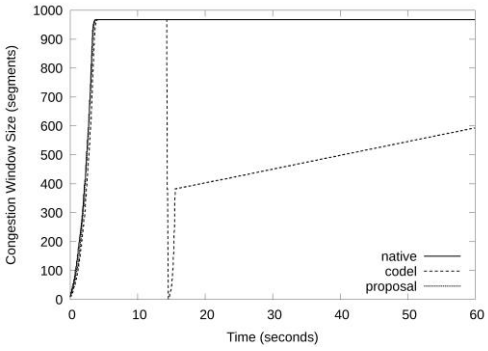
Figure 6. Results for TCP Reno with 100 msec additional delay.

V. CONCLUSIONS

This paper proposes a scheme for reducing the delay in TCP communication over 802.11n WLAN. Our scheme decreases the retransmission limit of the Block Ack based retransmission gradually according to the data rates becoming low. This paper also presents the detailed performance evaluation of our proposal, CoDel using the active queue management, and the native 802.11n with Cubic TCP and TCP Reno. The results show that our proposal and CoDel decrease the delay at a low data rate which the native 802.11n suffers from. The results also show that there are some cases where CoDel drops packets unnecessarily and the throughput in CoDel becomes lower at a high data rate. These results show that our proposal, which weakens the MAC level retransmission function can solve the bufferbloat problem specific for 802.11n WLAN.



(a) Throughput versus time



(b) Congestion window size versus time

Figure 7. Results for individual TCP Reno communications with 216 Mbps data rate (100 msec additional delay).

REFERENCES

[1] IEEE Standard for Information technology,"Local and metropolitan area networks Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," 2012.

[2] M. Nomoto, T. Kato, C. Wu, and S. Ohzahata, "Resolving Bufferbloat Problem in 802.11n WLAN by Weakening MAC Loss Recovery for TCP Stream," Proc.12th IASTED PDCN, pp. 293-300, Feb. 2014.

[3] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," ACM Queue, Virtualization, vol. 9, no.11, pp. 1-15, Nov. 2011.

[4] M. Allman, "Comments on Bufferbloat," ACM SIGCOMM Computer Communication Review, vol.43, no.1, pp. 31-37, Jan. 2013.

[5] A. Showail, K., Jamshaid, and B. Shihada, "An Empirical Evaluation of Bufferbloat in IEEE 802.11n Wireless Networks," Proc. IEEE WCNC '14, pp. 3088-3093, Apr. 2014.

[6] K. Nichols and V. Jacobson, "Controlling Queue Delay," ACM Queue, Networks, vol.10, no.5, pp. 1-15, May 2012.

[7] I. Rhee and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," SIGOPS Operating Systems Review, vol.42, no. 5, pp. 64-74, July 2008.

[8] IEEE Standard for Information technology, "Local and metropolitan area networks--Specific requirements--Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements," 2005.

[9] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Trans. On Networking, vol.1, no.4, pp. 397-413, Aug. 1993.

[10] 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, pp. 1465-1480, Oct. 1995.

[11] J. Bai, E. P. Eyisi, Y. Xue, and X. D. Koutsoukos, "Dynamic Tuning Retransmission Limit of IEEE 802.11 MAC Protocol for Networked Control Systems," Proc. 2010 IEEE/ACM Int. Conf. on Green Computing and Comunications, pp. 666-672, Dec. 2010.

[12] W. Wen and D. Liu, "An adaptive retry scheme for delay-constrained service transmission in 802.11n system," Proc. IEEE ICCP 2011, pp. 97-101, Oct. 2011.

[13] M. Kim and C. Choi, "Joint Rate and Fragment Size Adaptation in IEEE 802.11n Wireless LANs," Proc. 2011 IEEE CCNC, pp. 942-947, Jan. 2011.

[14] iperf, https://github.com/esnet/iperf, [retrieved: Nov. 2016].

[15] ath9k Linux Wireless, http://wireless.kernel.org/en/users/Drivers/ath9k, [retrieved: Nov. 2016].

[16] S. Hemminger, "Network Emulation with NetEm," Proc. 6th Australia's National Linux Conference (LCA2005), pp. 1-9, Apr. 2005.

[17] Linux foundation: tcpprobe, http://www.linuxfoundation.org/collaborate/workgroups/networking/tcpprobe, [retrieved: Nov. 2016].

 ISBN: 978-1-61208-546-3