A Solution for Bufferbloat Problem in Upload TCP Communication over IEEE 802.11n WLAN Only by Modifying Access Point

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 (WLANs) suffer from Bufferbloat problem such that the round-trip delay increases in upload Transmission Control Protocol (TCP) communications when the Media Access Control (MAC) level data rate is low such as 6.5 Mbps. This problem degrades user level quality of service for communications sharing the WLAN transmission queue. In order to resolve it, several methods are proposed including a method based on active queue management, our previous method reducing the MAC retransmission limit at low MAC level data rates, and a method stopping sending TCP level data when the queue contains more than a specific number of packets. However, those methods need to be implemented in data sending stations. In this paper, we propose a new method resolving Bufferbloat problem by providing an effect similar to our previous method at a data receiving access point. This decreases the congestion window size in the sending side TCP and can reduce the delay. This paper also shows the results of the performance evaluation by implementing the proposed method in an access point. According to the results, the proposed method decreases the Round-Trip Time (RTT) for low MAC data rates and does not reduce the TCP throughput. It does not give any influences on the TCP throughput for the stations supporting the conventional methods for Bufferbroat problem.**

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

I. INTRODUCTION

WLANs conforming to the IEEE 802.11n standard [1] have adopted new physical and MAC technologies. They include Multiple-Input and Multiple-Output (MIMO), the channel bonding, the frame aggregation, and the Block ACKnowledgment (Block ACK).

On the other hand, similarly with the conventional IEEE WLAN standards, IEEE 802.11n supports multiple data rates and the dynamic rate switching to use the optimal data rate between a STAtion (STA) and an Access Point (AP). When an STA 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 an STA 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.

When an STA communicates by TCP while it is using a low data rate, the packet losses do not increase, but the RTT increases largely, up to several seconds [2]. 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 this problem, there are some proposals [6]-[8]. All of them intend to decrease TCP traffic load when packets in a WLAN transmission queue pile up and the transfer delay increases. They are classified into two categories. One is a scheme that generates packet losses intentionally against increased transfer delay, which in turn decreases TCP congestion window (cwnd). CoDel [6], which is an active queue management based method, uses a packet-sojourn time in a transmission queue as a control parameter, and drops a packet in the situation when packets stay too long in the queue. In our previous paper [2], 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 a method that intentionally weakens the capability of retransmission realized by Block ACK frames, only when the data rate is low in TCP communications [7]. It increases the possibility of TCP packet losses at low data rate. The second category is a scheme that TCP stops sending data segments when many packets are stored in a MAC level transmission queue. An example is TCP small queues [8], which is implemented in the Linux operating system with 3.6 and later versions. For resolving Bufferbloat problem in upload TCP communications, all of those methods require to be implemented in every STA. The Linux operating system implements CoDel and/or TCP small queues, but as far as we know, the Windows and MAC operating systems do not implement either of them.

In this paper, we propose a new method which resolves Bufferbloat problem by providing an effect at a receiving side access point, which is similar to reducing the MAC level retransmission limit in STAs when the data rate is low. This decreases cwnd in the sending side and can reduce the delay. This paper also shows the results of the performance evaluation by implementing the proposed method in a Personal Computer (PC) based AP. According to the results, the proposed method decreases RTT for low MAC data rates and does not reduce the TCP throughput. It does not give any influences on the TCP throughput for STAs supporting CoDel or TCP small queues.

The rest of this paper is organized as follows. Section II explains the frame aggregation and Block ACK procedures and the related work. Section III describes the proposed method, and Section IV gives the results of performance evaluation. In the end, Section V concludes this paper.

II. PROCEDURE OF 802.11N AND RELATED WORK

This section describes the high throughput data transfer function of 802.11n and the related work on Bufferbloat problem.

 ISBN: 978-1-61208-596-8

A. Frame aggregation and Block ACK in 802.11n WLAN

IEEE 802.11n allows multiple data frames (called MAC Protocol Data Units: MPDUs) to be aggregated and sent together in order to increase the efficiency of data sending (see Figure 1). The whole transmitted frame is called Aggregation MPDU (A-MPDU), and is a collection of A-MPDU subframes, each of which includes an MPDU delimiter, and MPDU body, and a padding. An MPDU delimiter contains the MPDU length, a Cyclic Redundancy Check (CRC) to detect bit errors within the delimiter itself. A padding consists of 0 through 3 bytes, which makes the length of an A-MPDU subframe a multiple of 4 bytes.

The IEEE 802.11n standard adopts an acknowledgment scheme called High Throughput (HT)-immediate Block ACK. When a receiver receives an A-MPDU, it replies a Block ACK frame which contains a Block ACK Bitmap parameter indicating whether it correctly receives a MPDU with a specific sequence number. The Bitmap indicates receipt or non-receipt of 64 MPDUs. The data sender retransmits non-received MPDUs according to the Bitmap. When a Block ACK frame itself is lost, the whole A-MPDU is retransmitted by timeout.

These procedures are implemented by the WLAN device drivers and WLAN hardware at the data sender and receiver. The details are shown in Figure 2. At the data sender side, the device driver selects MPDUs to be aggregated in an A-MPDU. They include some retransmitted MPDUs determined by the Block ACK Bitmap. The device driver also determines the retry-out of MPDUs independently if the retransmission count of the MPDU reaches the retransmission limit. On the other hand, the WLAN hardware at the sender side realizes the actual formatting and sending out of A-MPDUs, timeout retransmission of A-MPDUs, and reporting of Block ACK Bitmap to the device driver.

At the data receiver side, the WLAN hardware handles the reporting of received MPDUs to the device driver and the sending of Block ACKs. The hardware also notifies the device driver of the information on MPDUs with CRC error. On the other hand, the device driver handles the sequencing of MPDUs and the retry-out decisions. The retry-out decision at the data receiver side is realized by the timeout basis, not by counting the MPDU retransmissions.

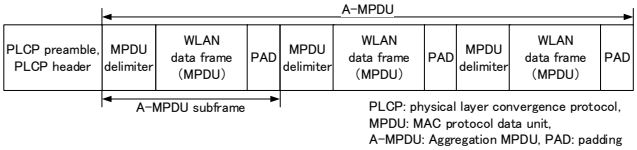


Figure 1. Format of A-MPDU.

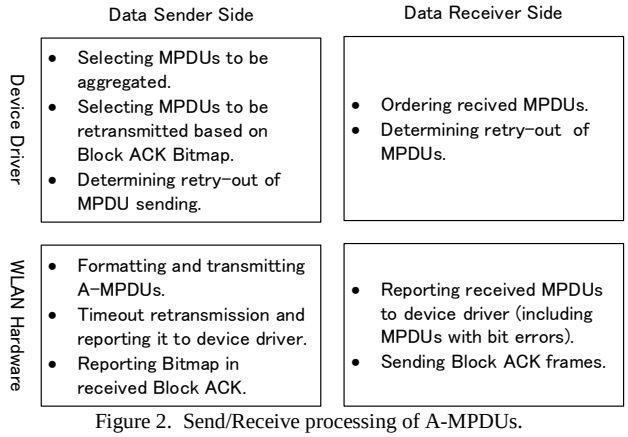


Figure 2. Send/Receive processing of A-MPDUs.

B. Related work

(1) CoDel

As described above, CoDel uses a 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]. The *interval* takes 100 msec at 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 started.

(2) Decreasing retransmission limit at low data rate

The second method is our previous proposal, which is based on the MAC level retransmission limit adjustment. It aims at causing an MPDU loss intentionally by setting the retransmission limit to some value between 2 and 8 when the data rate is smaller than 100 Mbps, and use 10, which is the default value, when the data rate is larger than 100 Mbps [7].

(3) TCP small queues

In an ordinary TCP communication, data that an application sent are stored in the send socket buffer and transferred under the limitation of advertised window and cwnd. In contrast, TCP small queues is a method that transfers TCP data segments by taking account of the status of transmission queues within the operating system scheduler and the WLAN device driver [8]. When the total size of TCP data segments stored in those transmission queues becomes a specific threshold (128 Kbytes in the default setting), new data generated by applications are not handled in TCP and keep staying in the send socket buffer. After the TCP data size stored in the transmission queue becomes smaller than the threshold, TCP module resumes the processing of data stored in the send socket buffer. As a result, TCP small queues can suppress the queuing delay without invoking intentional packet losses even if a low data rate is used.

As described above, all of those methods resolve Bufferbloat problem over 802.11n WLAN, but they need to be implemented in individual STAs. Although the recent versions of Linux implement CoDel and TCP small queues, it is expected that some of the major operating systems, such as Windows and Mac OS, support neither of them.

III. PROPOSAL

The primary goal of our proposal is to resolve Bufferbloat problem in upload TCP communications over 802.11n WLANs only by modifying an AP, which is an MPDU receiver. We adopt a method similar with our previous proposal, which increases MPDU losses by decreasing the MAC level retransmission limit when the data rate is low. The retransmission of lost MPDUs is controlled only by data senders; by STAs not APs in the upload data transfer. The

 ISBN: 978-1-61208-596-8

proposed method emulates the limited number of retransmissions only at an AP working as an MPDU receiver, for MPDUs including TCP segments transferred in a low data rate. This will increase the loss possibility of MPDUs and invoke the cwnd shrinking. This section describes detailed design of the proposed method.

A. Design principles

In order to design the proposed method, we have adopted the following principles. The first one is that we design the proposed method under the restriction to implement it inside the WLAN device driver. As described in the previous section, the retransmission of MPDUs and the sending of Block ACK are implemented in the WLAN hardware. So, the device driver at data receiver side cannot control the MPDU retransmission behavior that a data sender goes up to the retransmission limit. So, we adopt an approach that, when MAC level data rate is low, the receiver side device driver behaves as if retry-out occurred in response to a smaller retransmission count, and goes further to handling the following MPDUs. The retransmission count used as the retry-out limit is called a *retry-out index* in this paper.

The second principle is that we utilize the information reported from the WLAN hardware to the device driver as much as possible. As described above, the WLAN hardware reports the sequence number of an MPDU which suffers from transmission error (with CRC error). The proposed method uses this information as far as it maintains consistency with other information.

Figure 3 shows the idea of our proposal. STA and Access Point is connected through WLAN, and Access Point has an access to Ethernet. Figure 3 (a) shows the idea of our previous proposal [7]. The retransmission limit is set to 1 in STA. Frame 3 is transferred with CRC error twice, and so it is handled as a retry-out event. As a result, Access Point relays Frame 4 to Ethernet without sending Frame 3. Figure 3 (b) shows the idea of the method proposed in this paper. While STA has a large retransmission limit, Access Point supposes that it is 1. So, at the timing of the second erroneous reception (the first retransmission) of Frame 3, it is handled as if the retry-out happens (pseudo retry-out), and the next frame (Frame 4) is relayed to Ethernet. Even if Frame 3 is received correctly later, it is ignored.

B. Detailed design

The followings give the details on how to implement the proposed method.

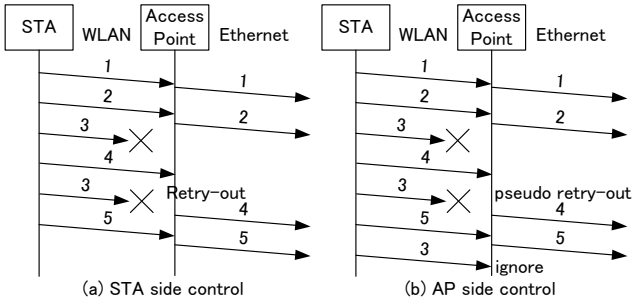


Figure 3. Idea of our proposal.

TABLE I. RETRY-OUT INDEX.

Smoothed data rate (Mbps)	～25	25～50	50～100	100～
Retry-out index	2	5	8	out of scope

(1) Retry-out index

The value of retry-out index is selected as shown in Table I. The value are the same as the retransmission limits used in our previous proposal [7]. The smoothed data rate used for the retry-out index selection is calculated according to the exponential moving average with coefficient 0.25. That is, the *smoothed data rate* in Table I is calculated by the following equation at each of MPDU reception.

$$smoothed\ data\ rate \leftarrow 0.75 \times smoothed\ data\ rate\ (previous) + 0.25 \times data\ rate\ used\ by\ received\ MPDU$$

(2) Estimation of retransmission count

An AP, an MPDU receiver, estimates the retransmission count per MPDU basis. The estimation is done based on the signaling of MPDUs with CRC error from the WLAN hardware. That is, when an MPDU with CRC error is reported, the proposed method increases (or set to 1) the estimated count of receiving this specific MPDU supposing that the signaled number of MPDU is correct. The retransmission count is the estimated receiving count minis one. It should be noted that the retransmission count covers both the Block ACK based MPDU retransmission and the timeout based A-MPDU retransmission.

(3) Handling as lost frames when reaching retry-out index

When an MPDU is received with CRC error and it contains a TCP segment, the estimated retransmission count is compared with the retry-out index. If the count reaches the retry-out index, the MPDU is marked as a lost frame. If there are any buffered MPDUs waiting for being reported to the upper layer whose sequence numbers follow the number of the MPDU marked as lost, they will be reported to the upper layer. Even though an MPDU is marked as lost, an STA continues the retransmission until its retransmission limit, and so an AP may receive such an MPDU.

(4) Non-error MPDU handling

When an AP receives an MPDU without any errors, it takes one of the followings. If the MPDU is marked as lost, it is ignored. (Note that a Block ACK is sent by the WLAN hardware.) Otherwise, if the MPDU is an in-sequence one, it is reported to the upper layer. Otherwise, it is buffered within the WLAN device driver for waiting in-sequence MPDUs.

C. Example of behavior at AP

Figure 4 shows an example of behavior at AP supporting the proposed method. The first row in the table indicates received A-MPDUs; A-MPDU(1) through A-MPDU(7). The second row indicates the sequence number assigned to individual MPDUs; 1 to 10. The slash mark over the number means that the corresponding MPDU is received with CRC error. For example, A-MPDU(1) includes five MPDUs, and among them, MPDUs with sequence number 2 and 4 are received with CRC error.

	A-MPDU (1)					A-MPDU (2)			A-MPDU (3)				A-MPDU (4)				(5)	(6)		(7)
	1	2	3	4	5	6	7	8	2	4	9	10	2	4	9	10	6	4	10	4
1	✔																			
2		0							1				✔							
3			Keep										✔							
4				0						1				2/Lost				3		Ignore
5					Keep									✔						
6						0											✔			
7							Keep										✔			
8								Keep									✔			
9											0				Keep		✔			
10												0				1			✔	

✔: Reporting received MPDU to upper layer
Keep: Buffering received MPDU within device driver
Lost: Handles MPDU as it is lost
Ignore: Ignoring received MPDU
Numbers: indicate retransmission counts

Figure 4. Example behavior of proposed method. .

Each cell in the table shows how the MPDU given in the second row is handled. The sequence number is indicated in the first column in the table. Non-blank cells mean the followings:

- Mark ✔ means the user data contained in the received MPDU is reported to the upper layer.
- "Keep" means that the received MPDU is buffered within the device driver.
- "Ignore" means that the user data in the received MPDU is ignored.
- "Lost" means that the received MPDU is marked as a lost frame.
- The number in a cell corresponds to the estimated retransmission count.

In this example, we suppose the retry-out index is two. In the beginning, A-MPDU(1) contains five MPDUs, among which MPDU-2 and 4 has CRC error. So, MPDU-1 is reported to the upper layer and MPDU-3 and 5 are buffered. For MPDU-2 and 4, the estimated retransmission count is set to 0.

Similarly, for A-MPDU(2), the estimated retransmission count of MPDU-6 is set to 0, and MPDU-7 and 8 are buffered.

A-MPDU(3) is an example of the case where a whole A-MPDU is corrupted. In this case, the information on MPDU sequence numbers is also reported to the device driver, which we have confirmed actually. As a result, the values of estimated retransmission count for MPDU-2, 4, 9 and 10 are incremented to 1 or set to 0, respectively.

A-MPDU(4) is a timeout based retransmission of A-MPDU(3), in which MPDU-2 and 9 are correctly received and MPDU-4 and 10 are corrupted. The receipt of MPDU-2 makes user data within MPDU-2 and 3 reported to the upper layer. MPDU-9 is buffered, and the estimated retransmission count for MPDU-4 and 10 is incremented. Since the count for MPDU-4 reaches 2 (retry-out index), MPDU-4 is handled as a lost frame at this time. Since buffered MPDU-5 follows the MPDU marked as lost (MPDU-4), its user data is reported to the upper layer.

Then, A-MPDU(5) containing only MPDU-6 is received. Since this is an in-sequence MPDU and MPDU-7 through 9 are buffered within the device driver, user data contained in all of those MPDUs are reported to the upper layer.

Next, A-MPDU(6) containing MPDU-4 and 10 is received, and MPDU-4 is again corrupted. Since MPDU-4 is marked as lost, its retransmission count is just incremented and user data in MPDU-10 is passed to the upper layer.

In the end, A-MPDU(7) containing MPDU-4 correctly is received. For MPDU-4, the WLAN hardware sends a Block ACK, but the device driver just ignores it.

In this way, an AP, an MPDU receiver, sets the retry-out index to the smaller value than the retransmission limit at an STA, an MPDU sender, and handles an MPDU as a lost frame earlier than an STA.

IV. Performance Evaluation

This section describes the results of experiments measuring the TCP throughput and the RTT of Ping (Internet Control Message Protocol (ICMP) Echo request/response) in the situation where Bufferbloat problem occurs. More specifically, the following points are examined.

- Whether the proposed method suppresses the increase of RTT or not. As for STA, we use PCs which run the Linux operating system, Windows and Mac OS.
- Whether MPDU losses introduced by the proposed method reduce the TCP throughput or not.
- Whether the proposed method gives any influence or not to the TCP throughput of STAs implementing CoDel or TCP small queues.

A. Experimental settings

Figure 5 shows the network configuration of our experiment. An STA 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, which provides just bridging in this experiment.

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 STA is located at various locations in the 1st and 2nd floors, and the stairs between them. The distance between the STA 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 STA is fixed and performs TCP and Ping communications for 60 seconds.

The AP is implemented using a Linux PC. For STA, we prepared a Linux PC, a Windows PC and a Mac PC. The specification of the AP and the STA is given in Table II. As for the Linux PC, we prepared three versions; a PC without any methods preventing Bufferbloat problem, a PC with TCP

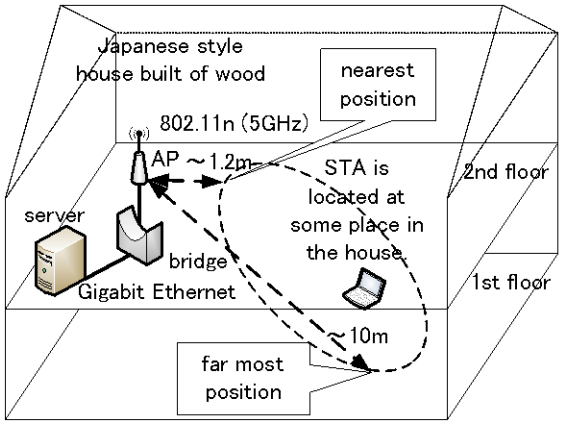


Figure 5. Experiment configuration.

TABLE II. SPECIFICATION OF AP AND PC.

	AP	Linux PC	Windows PC	Mac
model	Lenovo ThinkPad X61		Mac Book Pro	
kernel	Linux 3.2.38	Linux 3.13.0	Window 10	Mac OS 10.11
WLAN MAC	IEEE 802.11n (5GHz)			
TCP congestion control	—	CUBIC TCP [9]	Default method of OS	
AP software	Hostapd 0.7.3	—	—	—
NIC	NEC Aterm WL300NE-AG		Broadcom	
Driver	ath9k [10]		Default software of OS	

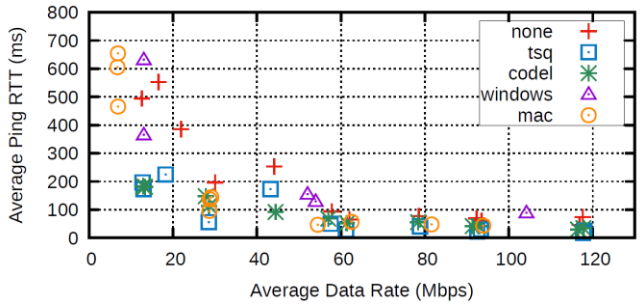
small queues, and a PC with CoDel. As for the PC with TCP small queues, we used a Linux PC with version 3.13 as it is. As for the PC without any methods, we used a Linux PC with version 3.13 by setting the queue limit to 10,000 packets, which is large enough to suppress the function of TCP small queues. As for the PC with CoDel, we used a Linux PC with large TCP small queues' limit by adding the function of CoDel.

B. *Results of performance evaluation*

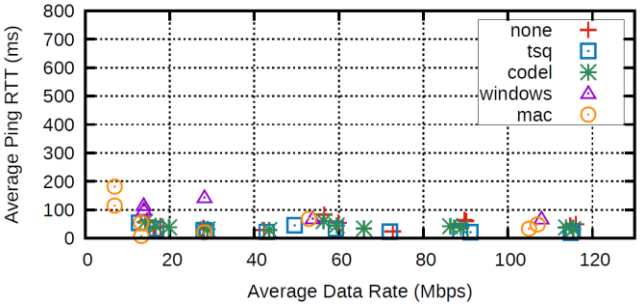
In the experiment, we executed a 60 sec TCP data transfer from STA to AP, and a 60 sec Ping communication invoked by STA, simultaneously. TCP data transfer is done by use of iperf [11]. In the Ping communication, STA sends a Ping request (ICMP Echo request packet) once a second. For one measurement run, we measured the average of TCP throughput, Ping RTT, and transmission queue length in WLAN driver. As for the last item, we measured only for a Linux PC.

As for the parameter which characterizes the position of the STA, we used the average data rate for all A-MPDUs transmitted during a 60 sec communication. We mapped the measured values with the average data rate. This is a similar approach with our previous paper [7].

Figure 6 shows the result of average Ping RTT versus average data rate. One plot in the graph corresponds an average during one measurement run. "none", "tsq" and



(a) Without proposed method in AP.



(b) With proposed method in AP.

Figure 6. Average Ping RTT vs. average data rate.

"codel" indicate the results of Linux PC without any methods, Linux PC with TCP small queues, and Linux PC with Codel, respectively. "windows" and "mac" indicate the results of Windows PC and Mac PC, respectively.

Figure 6 (a) shows that, in the case the proposed method is not introduced in the AP, the Ping RTT of Linux PC without any methods, Windows PC, and Mac PC becomes large when the average data rate is lower than 20 Mbps. The worst case is around 700 msec. In Linux PC with TCP small queues or CoDel, the Ping RTT is around 200 msec when the average data rate is lower than 20 Mbps.

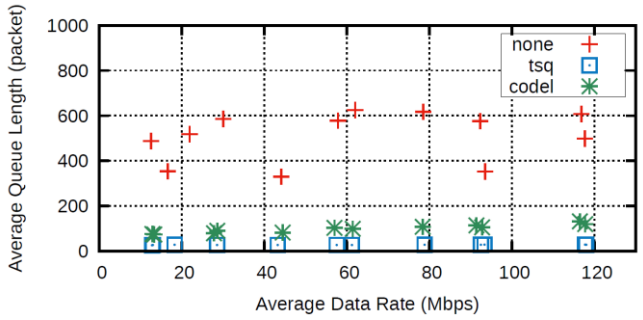
These results mean that Windows 10 and Mac OS X do not support any mechanisms to prevent Bufferbloat problem, and that TCP small queues and CoDel can suppress the increase of delay.

Figure 6 (b) shows that, when the proposed method is implemented in the AP, all of Linux PCs, Windows PC and Mac PC do not suffer from large delay even if the average data rate is lower than 20 Mbps. Especially, the reduction of Ping RTT is clear for Linux PC without any methods, Windows PC and Mac PC, which do not care about Bufferbloat problem. This result means that the proposed method is effective for preventing the increase of delay caused by Bufferbloat problem.

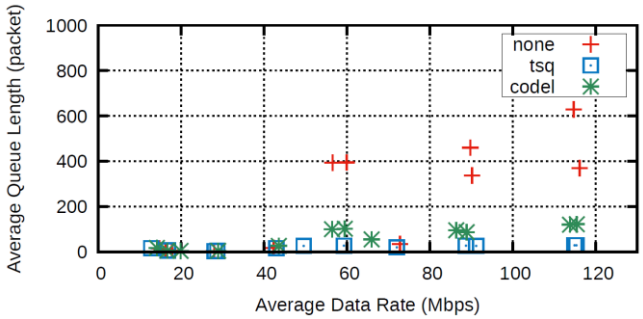
Figure 7 shows the result of average WLAN transmission queue length versus average data rate, in Linux PCs. In the case the proposed method is not introduced, around 600 packets are stored in transmission queue for any data rate in a Linux PC without any methods (Figure 7 (a)). This length is considered as cwnd in TCP communication [2]. On the other hand, in a Linux PC introducing TCP small queues or CoDel, the average queue length is reduced to around 100 packets, and as a result, the delay is reduced.

 ISBN: 978-1-61208-596-8

Figure 7 (b) shows that, when the proposed method is introduced, the average queue length in STA is suppressed to less than 50 packets when the average data rate is lower than 50 Mbps. As a result, the delay is also suppressed. When the average data rate is higher than 50 Mbps, the average queue length increases in a PC without any methods, but it is not a problem because the Ping RTT itself is small as shown in Figures 6 (a) and 6 (b).

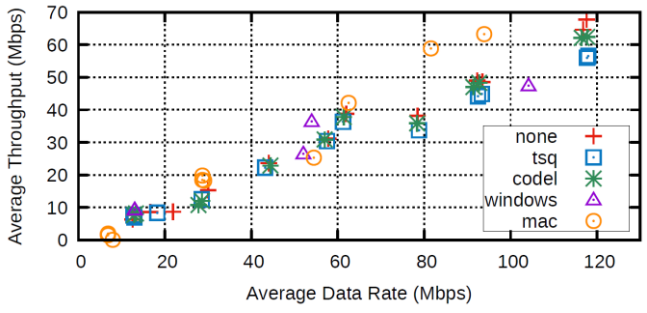


(a) Without proposed method in AP.

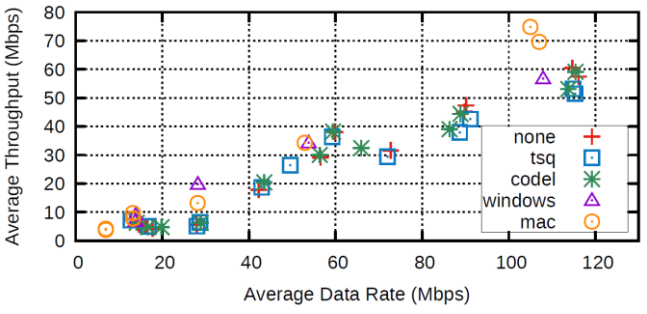


(b) With proposed method in AP.

Figure 7. Average transmission queue length vs. average data rate.



(a) Without proposed method in AP.



(b) With proposed method in AP.

Figure 8. Average TCP throughput vs. average data rate.

Figure 8 shows the result of average TCP throughput versus average data rate. As shown in Figure 8 (a), the TCP throughput is not affected by Bufferbloat problem. Besides, Figure 8 (b) shows that the result of TCP throughput is similar for both cases with and without the proposed method installed in the AP. This means that the proposed method does not influence the TCP throughput of STAs implementing TCP small queues or CoDel.

V. Conclusions

This paper has proposed a new method that is installed only at an IEEE 802.11n AP and can reduce the delay in upload TCP communications. The proposed method realizes similar effect with decreasing the retransmission limit at low MAC level data rate. The proposed method can prevent Bufferbloat problem in upload TCP communication without modifying sender side STAs at all. This paper also presented the detailed performance evaluation. The results clarified that the proposed method surely reduces the Ping RTT from STA when TCP bulk data transfer co-exists, that it does not reduce the TCP throughput, and that it does not give any influence to that of STAs implementing TCP small queues or CoDel, which are well-known methods for resolving Bufferbloat problem.

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] M. Nomoto, C. Wu, S. Ohzahata, and T. Kato, "Resolving Bufferbloat in TCP communication over IEEE 802.11n WLAN by Reducing MAC Retransmission Linit at Low Data Rate," in Proc. ICN 2017, pp. 69-74, Apr. 2017.

[8] E. Dumazet, "[PATCH v3 net-next] tcp: TCP Small Queues," Jul. 2012, http://article.gmame.org, [retrieved: Aug. 2017].

[9] 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.

[10] ath9k Linux Wireless, http://wireless.kernel.org/en/users/Drivers/ath9k, [retrieved: Aug. 2017].

[11] iperf, https://github.com/esnet/iperf, [retrieved: Aug. 2017].

 ISBN: 978-1-61208-596-8