Experimental Analysis and Resolution Proposal on Performance Degradation of TCP over IEEE 802.11n Wireless LAN Caused by Access Point Scanning

Toshihiko Kato, Kento Kobayashi, Sota Tasaki, Masataka Nomoto, Ryo Yamamoto, and Satoshi Ohzahata
Graduate School of Informatics and Engineering
University of Electro-Communications
Tokyo, Japan
kato@is.uec.ac.jp, kento_kobayashi@net.is.uec.ac.jp, t.souta@net.is.uec.ac.jp, noch@net.is.uec.ac.jp,
ryo_yamamoto@is.uec.ac.jp, ohzahata@is.uec.ac.jp

***Abstract*— In IEEE 802.11 wireless LAN, there is a problem that the access point scanning at stations which uses the power management function gives impacts on the performance of TCP communication. This paper is an extension of our previous paper that showed the result of experiments on this problem for uploading and downloading TCP data transfer over 802.11n wireless LAN. For the uploading transfer, we analyze the influence to TCP throughput focusing on the TCP small queues that limit the amount of data in wireless LAN's sending queue. We add some new results and discussions to the previous paper. As for the downloading transfer, we discuss the influence by the TCP congestion control algorithm at TCP senders and A-MPDU transmission rate at the access point. We also propose a method to stop access point scanning while data transfer is being performed, and show the results of the performance evaluation of the proposed method implemented on the Linux operating system.**

Keywords- WLAN; IEEE802.11n; Access Point Scanning; Power Management; TCP Small Queues; TCP Congestion Window Validation; NetworkManager; Linux Kernel Module.

I. INTRODUCTION

This paper is an extension of our previous paper [1] presented in an IARIA conference.

Nowadays, 802.11n [2] is one of most widely adopted IEEE wireless LAN (WLAN) standards. It establishes high speed data transfer using the higher data rate support (e.g., 300 Mbps), the frame aggregation in Aggregation MAC Protocol Data Unit (A-MPDU) and the Block Acknowledgment mechanism. On the other hand, TCP introduces some new functions to establish high performance over high speed WLANs. CoDel [3] and TCP small queues [4] that aim to resolve the Bufferbloat problem [5] are examples.

We reported some performance evaluation results on TCP behaviors over 802.11n [6][7]. While we were conducting those experiments, we encountered the situation that the TCP throughput decreases periodically. By analyzing the captured WLAN frame logs during the performance degradation, we confirmed that its reason is the periodical transmission of data frames without data (*Null data frames*) with the power management field set to 1 in the WLAN header. These frames are used by WLAN stations to inform access points that the stations are going to sleep and to ask the associated access point not to send data frames. It is pointed out that WLAN stations use Null data frames to scan another available access point periodically [8].

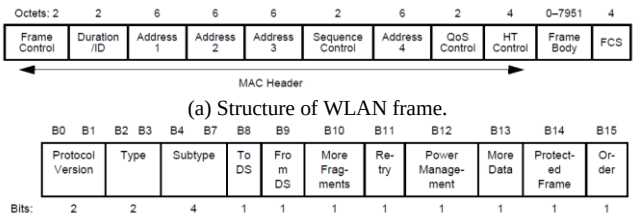
In our previous paper [1], we conducted detailed performance analysis and reported that the impacts of Null data frames on TCP data transfer change by the functions of TCP used in the communication. Specifically, the TCP sender behaviors and the throughput degradation depend on the direction of TCP data transfer (uploading from station to access point or downloading in the reverse direction), whether the TCP small queues are used or not, and what kind of congestion control algorithm is used. This paper is an extension of the previous paper, and shows the results of experimental analysis about the impacts on TCP throughput given by the access point scanning, by adding new results and discussions. This paper also proposes a method to stop access point scanning while data transfer is being performed. We implement the proposed method over *NetworkManager* software module running over the Linux operating system. This paper describes the implementation and performance evaluation of the proposed method.

The rest of this paper consists the following sections. Section II shows the technologies relevant to this paper. Section III explains the experimental settings. Sections IV and V give the detailed analysis of uploading TCP data transfer and downloading TCP data transfer together with access point scanning, respectively. Section VI proposes a method to protect the throughput degradation by the access point scanning. In the end, Section VII gives the conclusions of this paper.

II. RELEVANT TECHNOLOGIES

A. Power management function and Null data frames

As described above, IEEE 802.11 standards introduce the power management function. In the WLAN frame format depicted in Figure 1(a), bit 12 in the Frame Control field is the *Power Management field* (shown in Figure 1(b)). By setting this bit to 1, a station informs the associated access point that it is going to the *power save mode*, in which the station goes to sleep and wakes up only when the access point sends beacon frames. By setting the bit to 0, it informs the access point that it goes back to the *active mode*.



(a) Structure of WLAN frame.

(b) Structure of Frame Control field.

Figure 1. IEEE 802.11 WLAN frame format [2].

This function is used for several purposes. One example is the case that a station is actually going to sleep to save its power consumption for a while. In this case, a station wakes up only at the timing of receiving beacon frames from the access point. If the access point has data frames to deliver to the sleeping station, it indicates this fact in the Traffic Indication Map element in a beacon frame. In response to this information, the station requests the delivery of data frames by use of a PS-Poll frame.

Another example is the access point scanning. For example, a Lunix terminal executing the NetworkManager module searches periodically for an access point which provides stronger radio signal than the current access point with which the terminal is associated [9]. There are two schemes for the access point scanning; the passive scanning in which a station waits for a beacon frame from another access point, and the active scanning in which a station sends a probe request frame and waits for a probe response frame for the request. In either scheme, a station needs to ask the current access point to stop sending data frames to it. For this purpose, the station sends a frame with the Power Management field set to 1.

Null data frames are used by WLAN stations to inform access points of the shift to the power save mode or to the active mode [10]. This is a frame that contains no data (Frame Body in Fig. 1 (a)). An ordinary data frame has the type of '01' and the subtype of '0000' in the Frame Control field. That is, B3 and B2 in Fig. 1 (b) are 0 and 1, and B7 through B4 are all 0. On the other hand, Null data frame has the type of '01' and the subtype of '0100'. While an ordinary data frame has three Address fields, a Null data frame has only two Address fields; the transmitter is a station MAC address and the receiver is an access point MAC address. By using Null data frames with the Power Management field set to 0 or 1, stations can request the power management function for access points.

B. TCP small queues

In the Linux operating system with version 3.6 and later, a mechanism called TCP small queues [4] is installed in order to resolve the Bufferbloat problem. It keeps watching on the queues in Linux schedulers and device drivers in a sending terminal. If the amount of data stored in the queues is larger than the predefined queue limit, it suspends the TCP module until the amount of stored data becomes smaller than the limit. During this TCP suspension, the data which applications transmit is stored in the TCP send socket buffer, which may cause the application to be suspended if the TCP send socket buffer becomes full. After the TCP module is resumed, it processes the application data stored in the send socket buffer.

The default value of the predefined queue limit is 128 Kbyte, and it is adjustable by changing the following parameter;

/proc/sys/net/ipv4/tcp_limit_output_bytes.

This mechanism is different from the other mechanisms against the Bufferbloat problem, such as CoDel, in the point that no TCP segments are discarded intentionally to reduce TCP congestion window size.

C. Congestion window validation

The TCP congestion control uses the congestion window size (*cwnd*) maintained in TCP senders. TCP senders transmit TCP data segments under the limitation of cwnd and the window size advertised by TCP receivers. In general, cwnd is increased when TCP senders receive TCP acknowledgment (ACK) segments and is decreased when any data segments are retransmitted.

This mechanism is considered to work well under the assumption that the data transfer throughput is limited by cwnd. But it is possible that the throughput is controlled by an application in a TCP sender. In this case, the amount of data segments floating over network without being acknowledged (it is called *flight size*) might be smaller than cwnd. In an application limited case, however, cwnd also increases when the sender receives a new ACK segment, and the value of cwnd may be much larger than the current flight size. This means that the value of cwnd is invalid stage. If the TCP sender changes its status from application limited to cwnd limited suddenly, TCP segments corresponding to an invalid i.e. too large cwnd value will rush into a network.

In order to resolve such a problems, the congestion window validation (CWV) mechanism is proposed. RFC 2861 [11] proposes the following two rules. (1) A TCP sender halves the value of cwnd if no data segments are transmitted during a retransmission timeout period. (2) When a TCP sender does not send data segment more than cwnd during a round-trip time (RTT), then it decreases cwnd to

$$(cwnd + sent\ data\ size)/2$$

in the next RTT time frame.

RFC 7661 [12] revises the above rules and defines a new rule that, if the data size acknowledged during one RTT is smaller than half of cwnd, a TCP sender does not increase cwnd in the next RTT time frame. It defines the procedure in the case of congestion separately.

III. Experimental Settings

Figure 2 shows the configuration of the performance evaluation experiment we conducted. There are two stations conforming to 802.11n with 5GHz band and one access point connected to a server through 1Gbps Ethernet. One station called STA1 is associated with the access point, and communicates with the server through the access point. The other station called STA2 is used just to monitor WLAN frames exchanged between STA1 and the access point.

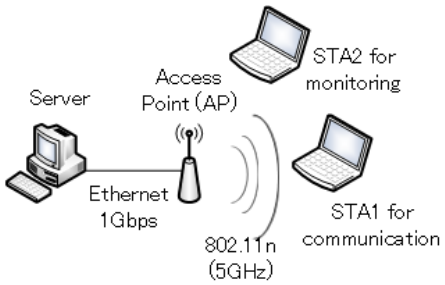


Figure 2. Network configuration in experiment.

TABLE I. SPECIFICATION OF NOTEBOOK AND ACCESS POINT.

NOTE	Manufacturer/Model	DELL Insilon 14
	Operating system	Ubuntu 14.04LTS (kernel 3.13) or Ubuntu 12.04LTS (kernel 3.2)
	WLAN driver	ath9k
AP	Manufacturer/Model	BUFFALO AirStation WZR-HP-AG300H
	WLAN chip	Atheros AR7161
	WLAN driver	ath9k

We use commercially available notebook PCs for the stations and the server. The access point used is also off-the-shelf product. The detailed specification of the notebook and the access point is shown in Table I. The access point is able to use the 40 MHz channel bandwidth and provides the MAC level data rate from 6.5 Mbps to 300 Mbps.

In the experiment, the data is generated by iperf [13] in the upload direction from STA1 to the server and the download direction from the server to STA1. The conditions for the experiment are the followings.

- Use or non-use of the TCP small queues, and
- use of CUBIC TCP [14] or TCP NewReno [15] as a congestion control mechanism.

When we use the TCP small queues, we installed Ubuntu 14.04 LTS in the notebook, and in the case not to use it, we installed Ubuntu 12.04 LTS.

During the data transmissions, the following detailed performance metrics are collected for the detailed analysis of the communication;

- the packet trace at the server, STA1 and STA2 , by use of *tcpdump*,
- the TCP throughput for every second, calculated from packet trace at TCP sender,
- the WLAN related metrics, such as the MAC level data rate, the number of A-MPDUs sent for a second and the number of MPDUs aggregated in an A-MPDU (an average during one second), from the device driver ath9k [16] at the access point and STA1, and
- the TCP congestion window size at the server, by use of *tcpprobe* [17] (an average during one second, calculated from the values obtained for every segment reception at the server).

IV. Analysis of Uploading TCP Data Transfer

In the experiments for uploading TCP data transfer, the results were different depending on whether the TCP small queues are used or not. On the other hand, the TCP congestion control algorithms did not affect the results so much. This section shows the results for uploading TCP data transfer focusing on the use or non-use of TCP small queues using CUBIC TCP.

A. Results when TCP small queues are used

Figure 3 shows the time variation of TCP throughput and cwnd, both of which are average value during one second, in the case that the TCP small queues are used in STA1. From this result, we can say that the throughput degradations occur periodically. Specifically, each *throughput degraded period* is around 10 sec. and such a period happens approximately once in 120 sec. In a *normal period*, the average TCP throughput is 136 Mbps, but it decreases to as much as 57 % in a throughput degraded period.

On the other hand, cwnd does not decrease even in a throughput degraded period, which means that there are no packet losses. Besides that, the increase of cwnd is depressed throughout the TCP communication. In this result, the value of cwnd is limited to around 200 packets.

Figure 4 shows the time variation of the MAC level data rate (average during one second). From this figure, it can be said that the data rate keeps high value. Figures 5 and 6 give

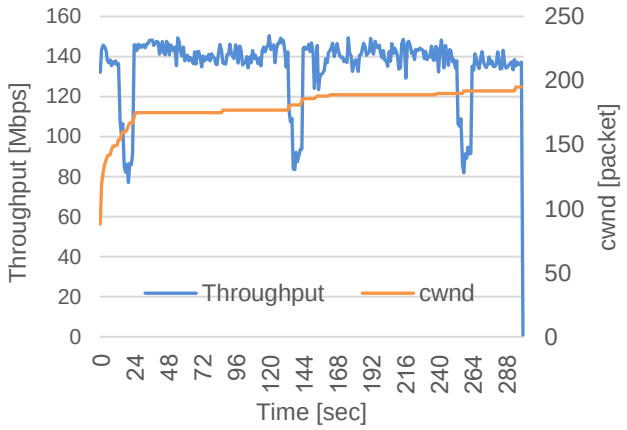


Figure 3. TCP throughput and cwnd vs. time in uploading data transfer with TCP small queues.

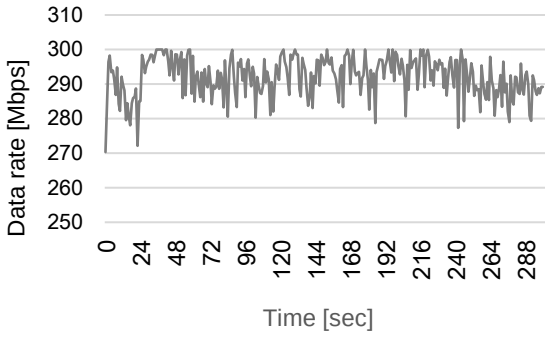


Figure 4. MAC level data rate vs. time in uploading data transfer with TCP small queues.

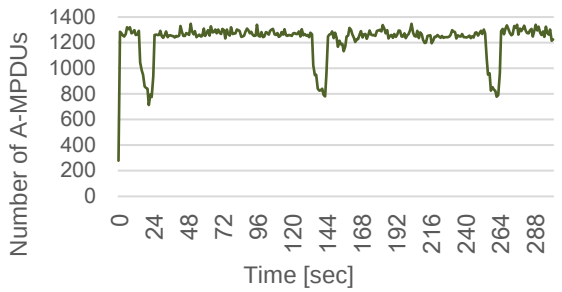


Figure 5. Number of A-MPDUs vs. time in uploading data transfer with TCP small queues.

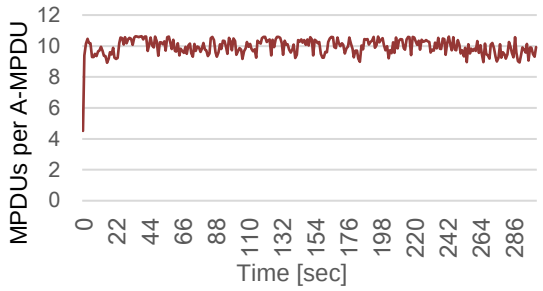


Figure 6. Number of A-MPDUs vs. time in uploading data transfer with TCP small queues.

the time variation of the number of A-MPDUs for one second and the number of MPDUs aggregated in an A-MPDU (an average during one second), respectively. Here, the number of MPDUs degrades during the throughput degraded period, while the number of MPDUs per A-MPDU keeps the same level. The decrease of the number of A-MPDUs is the reason for the periodic throughput degradation.

Figure 7 shows the packet trace, captured by STA2, of WLAN frames without data in the throughput degraded period starting at time 25 sec. This is the result of analysis by Wireshark and contains the number of frame, the time of packet capture measured from the TCP SYN segment, the source and destination MAC address, the protocol type (802.11), the frame length, and the information including name and parameters.

The frame whose number is 105,932, shown inverted to blue in the figure, is a Null data frame. For this frame, the Info column says that "Flags=. . .P. . .TC." This means that the Power Management field is set to 1 in this frame. The transmitter of this frame is STA1, whose MAC address is "LiteonTe_0b:ce:0c," and its receiver is the access point whose MAC address is "BuffaloI_27:2a:39." The sequence control ("SN" in the figure) is 1750 in this frame. In this packet trace, this Null data frame is not acknowledged by the access point. Instead, the same Null data frames are retransmitted eight times by the frames with numbers 105,933 through 105,953. They have the same sequence control (1750), and the Retry field in the Frame Control field is set to 1, as indicated "Flags=. . .PR. .TC" in the figure. From the fourth retransmission, a RTS frame is used before sending a Null data frame and the access point responds it by returning a CTS frame, which allows STA1 to send a Null data frame. But, from the fourth to the seventh retransmissions, the access point does not send any ACK frames. In the end, the access point sends an ACK frame at the eighth retransmission (see the frame with number 105,954). These frame exchanges takes 1.8 msec.

Then, the frame with number 105,955 is a beacon frame broadcasted by the access point. The duration between the ACK frame (No. 105,954) and this beacon frame is 90 msec in the figure.

24 msec after the beacon frame is transmitted, STA1

No.	Time	Source	Destination	Protocol	Length	Info
105931	25.814733	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	58	802.11 Block Ack, Flags=........C
105932	25.815296	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	57	Null function (No data), SN=1750, FN=0, Flags=...P...TC
105933	25.815318	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	57	Null function (No data), SN=1750, FN=0, Flags=...PR..TC
105934	25.815323	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	57	Null function (No data), SN=1750, FN=0, Flags=...PR..TC
105935	25.815326	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	57	Null function (No data), SN=1750, FN=0, Flags=...PR..TC
105938	25.815343	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	58	802.11 Block Ack, Flags=........C
105939	25.817033	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	46	Request-to-send, Flags=...P....C
105940	25.817054		LiteonTe_0b:ce:0c	802.11	40	Clear-to-send, Flags=........C
105941	25.817066	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	57	Null function (No data), SN=1750, FN=0, Flags=...PR..TC
105942	25.817070	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	46	Request-to-send, Flags=...P....C
105943	25.817073		LiteonTe_0b:ce:0c	802.11	40	Clear-to-send, Flags=........C
105944	25.817077	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	57	Null function (No data), SN=1750, FN=0, Flags=...PR..TC
105945	25.817080	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	46	Request-to-send, Flags=...P....C
105946	25.817083		LiteonTe_0b:ce:0c	802.11	40	Clear-to-send, Flags=........C
105947	25.817088	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	57	Null function (No data), SN=1750, FN=0, Flags=...PR..TC
105948	25.817091	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	46	Request-to-send, Flags=...P....C
105949	25.817095		LiteonTe_0b:ce:0c	802.11	40	Clear-to-send, Flags=........C
105950	25.817099	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	57	Null function (No data), SN=1750, FN=0, Flags=...PR..TC
105951	25.817102	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	46	Request-to-send, Flags=...P....C
105952	25.817105		LiteonTe_0b:ce:0c	802.11	40	Clear-to-send, Flags=........C
105953	25.817111	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	57	Null function (No data), SN=1750, FN=0, Flags=...PR..TC
105954	25.817116		LiteonTe_0b:ce:0c	802.11	40	Acknowledgement, Flags=........C
105955	25.907580	BuffaloI_27:2a:39	Broadcast	802.11	379	Beacon frame, SN=3924, FN=0, Flags=........C, BI=100, SSID=klab-n/a
105956	25.931649	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	57	Null function (No data), SN=1751, FN=0, Flags=.......TC
105958	25.931677		LiteonTe_0b:ce:0c	802.11	40	Acknowledgement, Flags=........C
105959	25.931682	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	57	Null function (No data), SN=1751, FN=0, Flags=....R..TC
105960	25.931686		LiteonTe_0b:ce:0c	802.11	40	Acknowledgement, Flags=........C
105961	25.931691	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	57	Null function (No data), SN=1751, FN=0, Flags=....R..TC
105962	25.932891	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	57	Null function (No data), SN=1751, FN=0, Flags=....R..TC
105963	25.932914	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	46	Request-to-send, Flags=........C
105964	25.932918		LiteonTe_0b:ce:0c	802.11	40	Clear-to-send, Flags=........C
105965	25.932922	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	57	Null function (No data), SN=1751, FN=0, Flags=....R..TC
105966	25.932927		LiteonTe_0b:ce:0c	802.11	40	Acknowledgement, Flags=........C
105967	25.932937	LiteonTe_0b:ce:0c	BuffaloI_27:2a:39	802.11	54	Null function (No data), SN=1752, FN=0, Flags=.......TC
105968	25.932940		LiteonTe_0b:ce:0c	802.11	40	Acknowledgement, Flags=........C

Figure 7. An example of packet trace during throughput degraded period focusing on WLAN frames without data

sends a Null data frame with the Power Management field set to 0, whose number and sequence control is 105,956 and 1751, respectively. Again, this Null data frame is retransmitted four times. In this case, although STA2 captures the corresponding ACK frames from the access point, STA1 retransmits it, repeatedly. After this frame is acknowledged by the ACK frame with number 105,966, STA1 transmit the next Null data frame whose sequence control 1752, and it is immediately acknowledged. These frame exchanges take 1.3 msec.

During this sequence, STA1 and the access point do not send any data frames. This paper refers to the time period as a *sleeping period*. After the last Null data frame with the Power Management field set to 0 is acknowledged, the data transfer from STA1 is restarted. But, several hundred milliseconds later, the similar communication sequence including the Null data frames and beacon frame occurs, and it goes to another sleeping period. This paper refers to the period when data frames are transmitted between sleeping periods as an *awoken period*. During a performance degraded period, there are around 26 pairs of sleeping period and awoken period.

The relationship among those periods are shown in Figure 8. In the evaluation result, the throughput degraded periods and the normal periods are repeated as shown at the top of this figure. The average duration for them is around 10 sec. and 110 sec., respectively. A throughput degraded period consists of sleeping periods and awoken periods. The average duration for them is 110 msec and 320 msec, respectively. As described in Figure 7 before, a sleeping period consists of a period sending Null data frames with the Power Management field set to 1, a period waiting for a beacon frame, and a period sending Null data frames with the Power Management field set to 0. The average durations for the individual periods are shown in Figure 8.

On the other hand, Figure 3 shows that the increase of cwnd is suppressed even if there are no packet losses. The reason is considered to be the collaboration of the TCP small queues and CWV. As described before, CWV intends to be used when a TCP communication is application limited. In the case the TCP small queues are used, however, it is possible that the data transfer stops when the buffered data in the sending queue for WLAN device exceeds the predefined queue limit, even if the flight size is smaller than cwnd. In this case, the MAC level data rate dominates the throughput instead of cwnd in the TCP level, and therefore, the control by CWV becomes effective. Actually, the source program of the TCP small queues implements a procedure such that, when the unacknowledged data amount is smaller than the current value of cwnd in the slow start phase, cwnd is not incremented even if a new ACK segment arrives. Note that this procedure itself is not conforming to RFC 2861 strictly but similar to RFC 7661, which was not standardized when the TCP small queues were introduced.

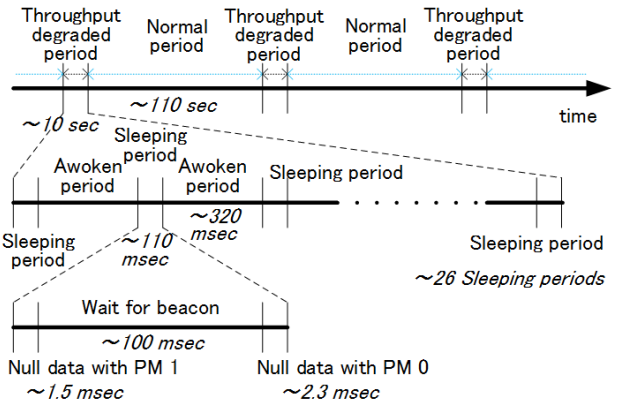


Figure 8. Detailed analysis of time periods.

B. Results when TCP small queues are not used

Figure 9 shows the time variation of TCP throughput and cwnd in the case that the TCP small queues are not used in STA1. In this case, the throughput degradations also occur periodically. In the normal periods, the average throughput is 174 Mbps, which is 38 Mbps higher than the case using the TCP small queues. This result seems to come from the fact that the cwnd value goes up to 970 packets. On the other hand, in the throughput degraded periods, the throughput decreases as low as 10 % of that in the normal period. The value of cwnd decreases largely in the throughput degraded periods.

Figure 10 shows the time variation of the MAC level data rate. In comparison with Figure 4, there are some drops of data rate at the timing of the throughput drop, but the drop is from 300 Mbps to around 280 Mbps, so it can be said that

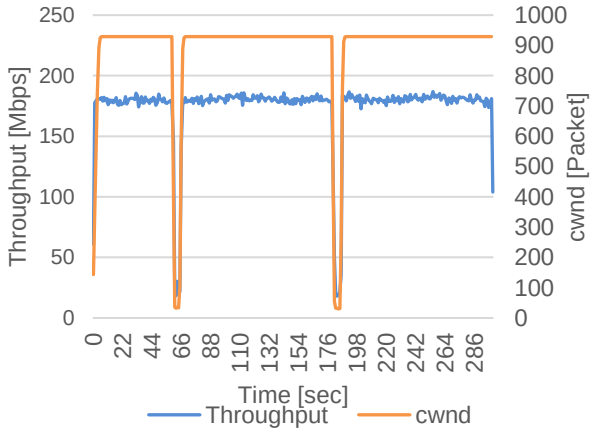


Figure 9. TCP throughput and cwnd vs. time in uploading data transfer without TCP small queues.

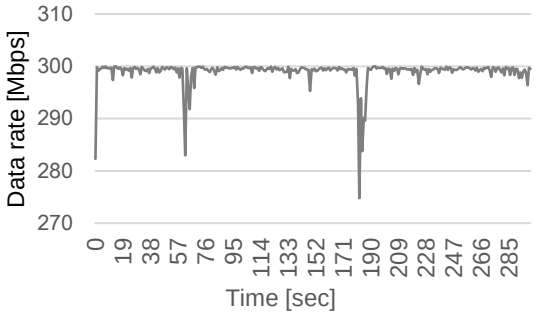


Figure 10. MAC level data rate vs. time in uploading data transfer without TCP small queues.

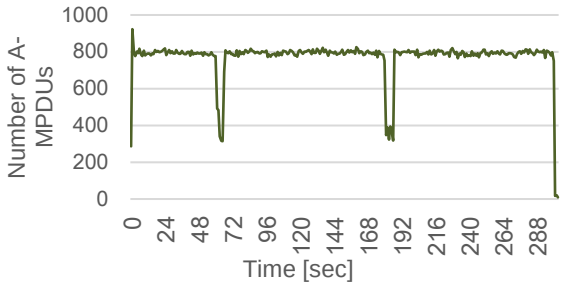


Figure 11. Number of A-MPDUs vs. time in uploading data transfer without TCP small queues.

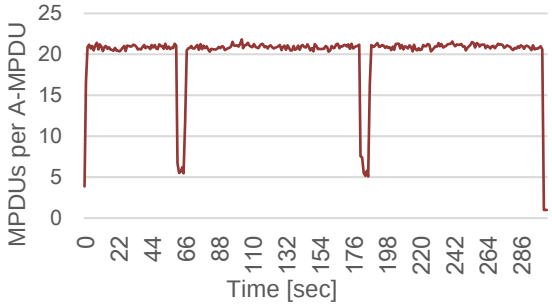


Figure 12. Number of A-MPDUs vs. time in uploading data transfer without TCP small queues.

the drop itself is not large. Figures 11 and 12 show the time variation of the number of A-MPDUs for one second and the number of MPDUs aggregated in an A-MPDU, respectively. In the case that the TCP small queues are used, only the number of MPDUs in an A-MPDU decreases in the throughput degraded periods. However, when the TCP small queues are not used, the number of MPDUs also decreases in the throughput degraded periods.

In this case, the reason for the periodic throughput degradation is also the periodic access point scanning using Null data frames with the power management function. In this case, however, there are some differences compared with the case of using the TCP small queues. At first, in the normal periods, the throughput and cwnd have larger values. On the other hand, in the throughput degraded periods, the drop of throughput is sharp, and cwnd as well as the number of MPDUs in one second drop sharply.

The reason is that there are some packet losses in the throughput degraded periods. The detailed discussions on the difference between the use and non-use of the TCP small queues are given in the following subsection.

C. Discussions

Figure 13 shows how the internal modules behave during a throughput degraded period, when the TCP small queues are used. During a sleeping period within a throughput degraded period, the WLAN module (WLAN interface hardware and its device driver) stops sending data requested from the IP module. As a result, several data are stored in the send queue maintained by the operating system and the driver. If the number of stored data exceeds the threshold, the TCP module stops reading data from the APP, which means application, module and they are kept in the send

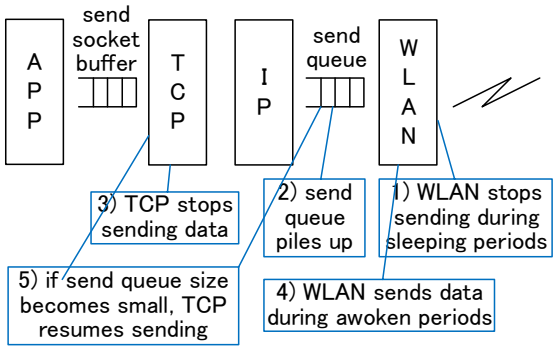


Figure 13. Behaviors when TCP small queues are used.

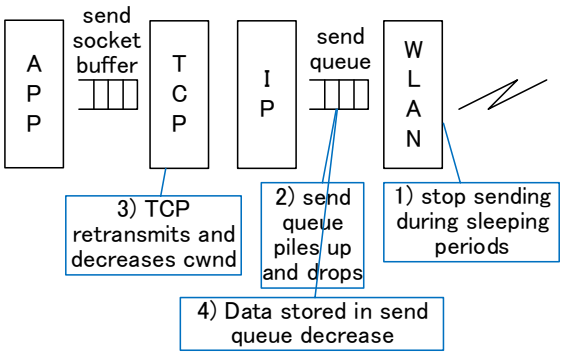


Figure 14. Behaviors when TCP small queues are not used.

socket buffer. This is the function of the TCP small queues. If the send socket buffer becomes full, the APP module stops sending data. In this situation, the WLAN module just keeps sending data stored in the send queue during the awoken periods, which is the same as the normal data transfer situation. If the size of data stored in the send queue is smaller the threshold, the TPC module resumes the sending. Therefore, only the TCP throughput and the number of A-MPDUs sent in one second decrease as shown in Figures 3 through 6.

Figure 14 shows the behaviors of the internal modules during a throughput degraded period, when the TCP small queues are not used. During a sleeping period within a throughput degraded period, the WLAN module stops sending data requested from the IP module. As a result, several data are stored in the send queue. This is the same as above. In this case, however, cwnd in the TCP module keeps increasing for every ACK segment while data segments are not lost. In the results in Figure 9, cwnd goes up to 970 packets as described above. During a sleeping period, the WLAN module stops sending data, but the TCP module keeps transmitting data, and so it is possible that the send queue overflows and some data segments are discarded. The TCP module retransmits lost data and decreases cwnd. As a result, the size of data stored in the send queue is also reduced. This means that the traffic load to the WLAN module is reduced and, in the awoken periods, the WLAN traffic is reduced. This corresponds to the drop of the TCP throughput, the number of A-MPDUs sent, and the number of MPDUs aggregated in an A-MPDU, as shown in Figures 9 through 12.

V. Analysis of Downloading TCP Data Transfer

In the experiments for downloading TCP data transfer, the results were not different depending on the use or non-use of TCP small queues. This is because neither the TCP small queues nor CWV are implemented at the access point. Instead, the results slightly depended on the TCP congestion control algorithms in the server. This section shows these results.

A. Results when CUBIC TCP is used

Figure 15 shows the time variation of TCP throughput and cwnd in the case that CUBIC TCP is used at the server. From this figure, we can say that the periodic throughput degradation also occurs at the downloading TCP data transfer. By analyzing the monitoring results of WLAN frames captured by STA2, we confirmed that there are periodic exchanges of Null data frames and beacon frames between STA1 and the access point, which is similar with the sequence in the uploading TCP data transfer. So, in the case of downloading TCP data transfer, the access point scanning by WLAN stations reduces the throughput.

As shown in the figure, cwnd at the server takes the value between 350 and 500 packets. The drops of cwnd indicate that packet losses occur frequently. The increase of cwnd takes a cubic curve of time, which is characteristic for CUBIC TCP.

Figure 16 shows the time variation of the MAC level data rate. Mostly the MAC data rate of 270 Mbps is kept. There is one drop, but the rate is still as high as 240 Mbps. It can be said that the MAC level data rate maintains a high level.

In contrary to the upload results, the throughput in a throughput degraded period drops sharply, although the cwnd value does not decrease largely during this period. In addition, the throughput just after a throughput degraded period is rather low. In order to investigate those results, we checked the number of A-MPDUs sent in one second by the access point, and the number of MPDUs contained in one A-MPDU. The results are given in Figures 17 and 18. These figures show that both A-MPDUs and MPDUs per A-MPDU decrease largely in throughput degraded periods. This result accounts for the throughput reduction. Besides that, the number of MPDUs aggregated in an A-MPDU is low just after a throughput degraded period. This is considered the reason for low throughput in this time frame.

B. Results when TCP NewReno is used

Figure 19 shows the time variation of TCP throughput and cwnd in the case that TCP NewReno is used at the server. Figure 20 shows the time variation of the MAC level data rate. Figures 21 and 22 show the time variation of the number of A-MPDUs sent in one second by the access point, and the number of MPDUs contained in one A-MPDU, respectively.

From Figure 19, it is confirmed that the throughput is degraded sharply in every 120 sec. From the monitoring results of WLAN frames captured by STA2, we also confirmed the periodic exchanges of Null data frames and beacon frames between STA1 and the access point. This is an access point scanning by STA1 and the reason for the

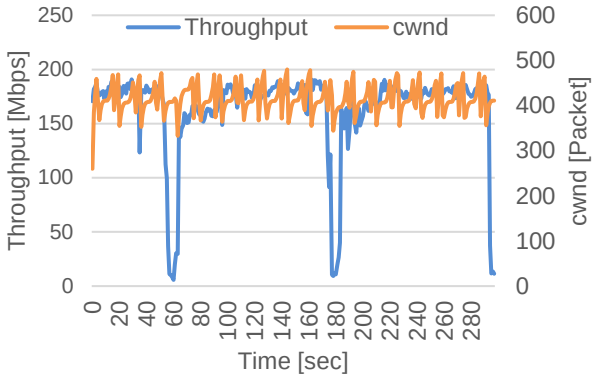


Figure 15. TCP throughput and cwnd vs. time in downloading data transfer using CUBIC TCP at server.

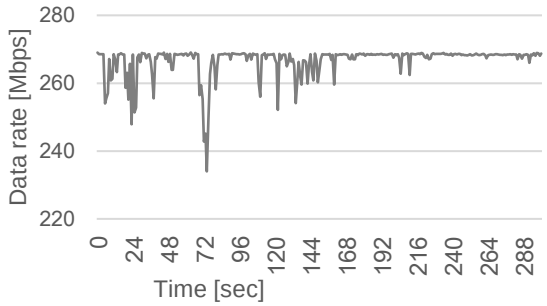


Figure 16. MAC level data rate vs. time in downloading data transfer using CUBIC TCP at server.

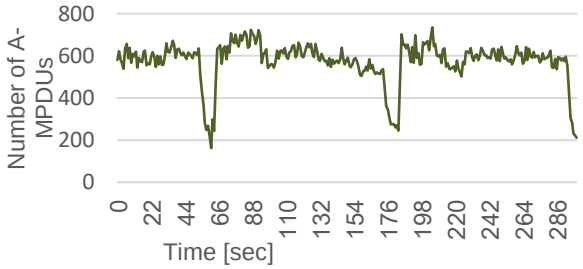


Figure 17. Number of A-MPTU vs. time in downloading data transfer using CUBIC TCP at server.

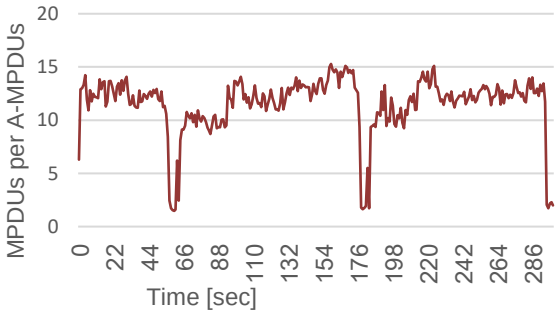


Figure 18. Number of MPTUs in A-MPDU vs. time in downloading data transfer using CUBIC TCP at server.

throughput reduction. This figure also shows that cwnd at the server takes the value between 300 and 500 packets, and that cwnd drops frequently, similarly with the case of CUBIC TCP. The increase of cwnd takes a linear curve along with time, which is characteristic for TCP NewReno.

From Figure 20, it can be said that, although there some decreases, the MAC level data rate keeps high value such as 270 Mbps.

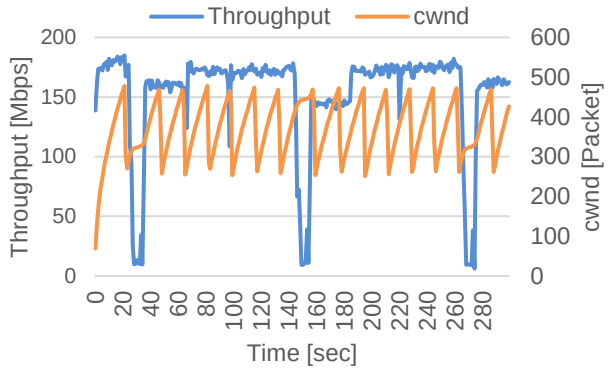


Figure 19. TCP throughput and cwnd vs. time in downloading data transfer using TCP NewReno at server.

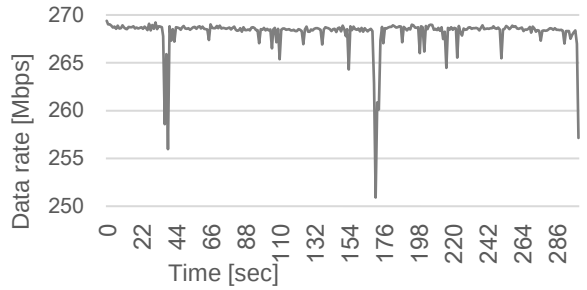


Figure 20. MAC level data rate vs. time in downloading data transfer using TCP NewReno at server.

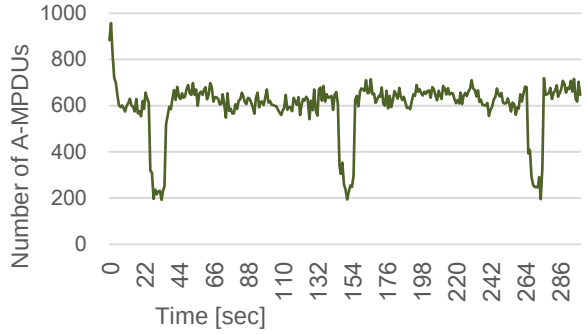


Figure 21. Number of A-MPTU vs. time in downloading data transfer using TCP NewReno at server.

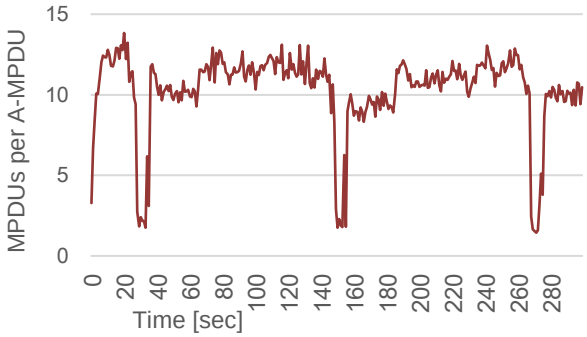


Figure 22. Number of MPTUs in A-MPDU vs. time in downloading data transfer using TCP NewReno at server.

While the throughput in a throughput degraded period drops sharply, the cwnd value does not decrease largely. The throughput just after a throughput degraded period is rather low. These are similar with the case of CUBIC TCP. As Figures 21 and 22 show, the numbers of transmitted A-MPDUs and MPDUs per A-MPDU decrease largely in throughput degraded periods. Besides that, the number of MPDUs aggregated in an A-MPDU is low just after a throughput degraded period. This will be the reason for low throughput in this time frame. These results are similar with the case of CUBIC TCP.

VI. Method to Stop Access Point Scanning

In this section, we present a method to avoid the throughput degradation by stopping access point scanning while data transfer is being performed.

A. Implementation

As we mentioned above, the access point scanning is realized by the NetworkManager software module in the Linux operating system [18]. It is executed as a daemon to support network configuration and operation. The software of NetworkManager is maintained within the Linux package management system that maintains binary files of various software, configuration files, and the information about their dependencies. Since the Ubuntu distribution we use in this paper depends on the Debian package management system, based on a tool called `dpkg` with the `apt` (advanced packaging tool) system.

Figure 23 shows a Linux script that allows a modified NetworkManager source program to be installed within the Debian package management system [19]. In the first step, the binary NeworkManager is installed using an `apt-get install` command. Then, some software necessary for package building is installed in the second step. After that, the source program of NetworkManager is downloaded under the subdirectory `net-man` using an `apt-get source` command. By this step, several c/h files are expanded under the directory `network-manager-0.9.4.0`. The number `0.9.4.0` indicates the version of NetworkManager, which corresponds to Ubuntu 14.04 LTS. The source files are expanded in the `src` directory under `network-manager-0.9.4.0`. At this timing, the downloaded software are modified as described below. This is step 4. Step 5 is preparing the package building, and the `debuild` command builds the NetworkManager package. By these steps, there are several `deb` files generated. In the end, a `dpkg` command generates an executable file for NetworkManager.

The access point scanning is realized by the C language functions described in Figure 24.

- In the function `schedule_scan()`, the function `request_wireless_scan()` is called periodically by use of `g_timeout_add_seconds()`.
- The function `g_timeout_add_seconds()` is a function to make another function called at regular intervals, which is defined in the framework of GTK+ (the GIMP Toolkit) [20].

```
# step 1: install binary NetworkManager
sudo apt-get install network-manager
sudo apt-get update

# step 2: prepare package building
sudo apt-get install dpkg-dev devscripts fakeroot

# step 3: get NetworkManager source
mkdir src
cd src
apt-get source network-manager

# step 4: modify NetworkManager source

# step 5: install build package
sudo apt-get build-dep network-manager

# step 6: build NetworkManager
cd network-manager-0.9.4.0
debuild -uc -us -b

# step 7: install modified NetworkManager
sudo dpkg -i ../*.deb
```

Figure 23. Linux script to build NetworkManager from source program.

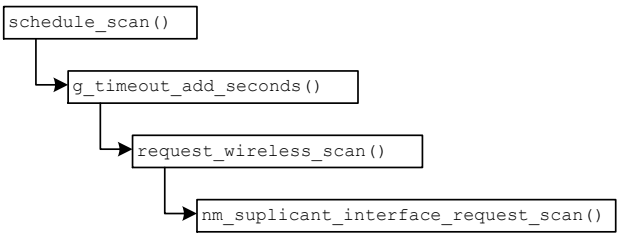


Figure 24. Functions for access point scanning.

- The function `request_wireless_scan()` calls `nm_suplicant_interface_request_scan()`, which performs access point scanning actually.

Based on this consideration, we modified the function `request_wireless_scan()` in the way that, if some data transfer has been performed since the last scan request, the actual scanning is skipped.

B. Performance evaluation

We evaluate the TCP throughput of the proposed method. The evaluation is done in the same configuration described in Section III. We use the uploading data transfer with the TCP small queues and CUBIC TCP. Figures 25 and 26 show the time variation of TCP throughput, which is an average during one second, without and with the proposed method implemented in a station, respectively. As shown in Figure 25, the TCP throughput is degraded around every 120 second, when the proposed method is not implemented. This is similar with the result in Figure 3. On the other hand, Figure 26 shows that, when the proposed method is implemented in a station, these periodic large drops of TCP throughput do not occur. These results indicate that the proposed method can resolve the throughput degradation problem caused by the access point scanning.

VII. CONCLUSIONS

This paper discussed the results on performance evaluation on the periodic TCP throughput degradation in IEEE 802.11n WLAN. The degradation is invoked by the periodic access point scanning using Null data frames with the Power Management field set to on and off. We showed that the throughput degradation is different depending on whether the TCP small queues are used or not in an uploading TCP data transfer, and what type of TCP congestion control algorithms are used in a downloading TCP data transfer. In the uploading data transfer, the TCP small queues and the congestion window validation suppress both of the increase of congestion window size in a normal period and its decrease in a throughput degradation period. So, the throughput degradation invoked by the access point scanning is smaller than the case when the TCP small queues are not used. In the downloading data transfer, the congestion control algorithms give some impacts on the time variation of congestion window size. However, the actual reason for the throughput degradation is the decrease in the transmission rate of A-MPDUs and the number of MPDUs in one A-MPDU, which are observed at an access point. This paper also proposed a method to resolve the performance degradation caused by access point scanning by stopping scanning while data transfer is being done. The proposed method was implemented in the NetworkManager software module running over the Linux operating system, and the performance evaluation showed that the performance degradation is resolved by the proposed method.

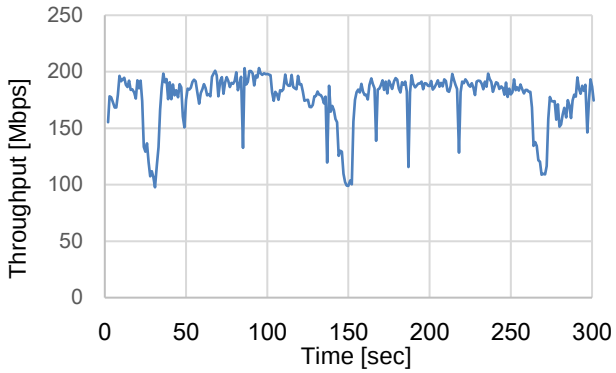


Figure 25. TCP throughput vs. time in uploading data transfer without proposed method.

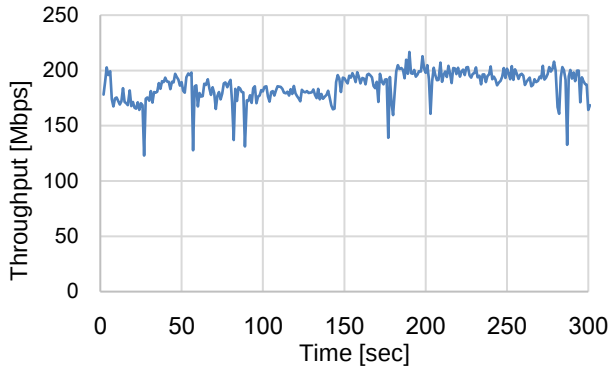


Figure 26. TCP throughput vs. time in uploading data transfer with proposed method.

REFERENCES

[1] K. Kobayashi, Y. Hashimoto, M. Nomoto, R. Yamamoto, S. Ohzahata, and T. Kato, "Experimental Analysis on Access Point Scanning Impacts on TCP Throughput over IEEE 802.11n Wireless LAN," Proc. ICWMC 2016, pp. 115-120. Nov. 2016.

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

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

[4] Eric Dumazet, "[PATCHv2 net-next] tcp: TCP Small Queues," http://article.gmane.org/gmane.network.routing.codel/68, 2012, retrieved Feb. 2018.

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

[6] 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. PDCN 2014, pp. 293-300, Feb. 2014.

[7] Y. Hashimoto, M. Nomoto, C. Wu, S. Ohzahata, and T. Kato, "Experimental Analysis on Performance Anomaly for Download Data Transfer at IEEE 802.11n Wireless LAN," Proc. ICN 2016, pp. 22-27, Feb. 2016.

[8] My80211.com, "802.11: Null Data Frames," http://www.my80211.com/home/2009/12/5/80211-null-data-frames.html, Dec. 2009, retrieved Feb. 2018.

[9] ath9k-devel@lists.ath9k.org, "disable dynamic power save in AR9280," http://comments.gmane.org/gmane.linux.drivers.ath9k.devel/5199, Jan. 2011, retrieved Feb. 2018.

[10] W. Gu, Z. Yang, D. Xuan, and W. Jia, "Null Data Frame: A Double-Edged Sword in IEEE 802.11 WLANs," IEEE Trans. Parallel & Distributed Systems, vol. 21, no. 7, pp. 897-910, Jul. 2010.

[11] N, Handley, J. Padhye, and S. Floyd, "TCP Congestion Window Validation," IETF RFC 2861, Jun. 2000.

[12] G. Fairhurst, A. Sathiaseelan, and R. Secchi, "Updating TCP to Support Rate-Limited Traffic," IETF RFC 7661, Oct. 2015.

[13] iperf, http://iperf.sourceforge.net/, retrieved Feb. 2018.

[14] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," ACM SIGOPS Operating Systems Review, vol. 42, no. 5, pp. 64-74, July 2008.

[15] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm," IETF RFC 3728, April 2004.

[16] ath9k Linux Wireless, https://wireless.wiki.kernel.org/en/users/drivers/ath9k, retrieved Feb. 2018.

[17] Linux foundation: tcpprobe, http://www.linuxfoundation.org/collaborate/workgroups/networking/tcpprobe, retrieved Feb. 2018.

[18] ubuntu documentation, "NetworkManager," https://help.ubuntu.com/community/NetworkManager, retrieved Feb. 2018.

[19] Devian/Wiki/, "Building Turorial," https://wiki.debian.org/BuildingTutorial, retrieved Feb. 2018.

[20] The GTK+ Project, "What is GTK+, and how can I use it?" https://www.gtk.org/, retrieved Feb. 2018.