Management Studies, May-June 2017, Vol. 5, No. 3, 219-233
doi: 10.17265/2328-2185/2017.03.007

Controlling Middleware for Reducing the TCP ACK Packet Backlog at the WLAN Access Point

Ai Hayakawa, Masato Oguchi
Ochanomizu University, Tokyo, Japan
Saneyasu Yamaguchi
Kogakuin University, Tokyo, Japan

Although modern loss-based Transmission Control Protocols take aggressive congestion window (CWND) control strategies in order to gain better throughput, such strategies may cause a large number of packets to be backlogged and eventually dropped at the entry point to the wireless access network. This problem applies not only to the downstream TCP sessions but also to the upstream TCP sessions when the terminal is connected via a Wireless Local Area Network (WLAN), which disregards the size of packets in its scheduling. This paper focuses on the ACK packet backlog problem with the upstream TCP sessions, and proposes a CUBIC based CWND control mechanism as part of the middleware for the Android terminals. It utilizes the Round Trip Time (RTT) as an indication for the TCP ACK backlog condition at the WLAN AP, and controls the upper and lower bounds of its CWND size to suppress excessive transmissions of own TCP DATA packets. Unlike the existing works, the proposed mechanism is based on WLAN CSMA/CA performance studies and then it does not require manual performance tuning. An experimental study with up to 10 Android terminals shows that the proposed mechanism can improve both aggregate throughput and fairness of the WLAN, and that it is highly effective particularly for cases where very long RTTs are observed.

Keywords: TCP, WLAN, Congestion control, RTT, Android

Introduction

Modern loss-based TCPs such as BIC (Xu, Harfoush, & Rhee, 2003) and CUBIC (Ha, Rhee, & Xu, 2008) adopt aggressive CWND control strategies to gain better throughput over other competing TCP sessions. However, such strategies may cause a large number of packets to be backlogged and eventually dropped at the entry point to the wireless access network because a wireless link can usually offer narrower bandwidths than its wired backhaul and backbone networks. This problem applies not only to the downstream TCP sessions but also to the upstream TCP sessions when the terminal is connected via a WLAN, which disregards the size of packets in its CSMA/CA (Cuina, Jing, & Lianfeng, 2010) based on scheduling.

Ai Hayakawa, M. S., Ochanomizu University, Tokyo, Japan.
Saneyasu Yamaguchi, Ph.D., associate professor, Ochanomizu University, Tokyo, Japan.
Masato Oguchi, Ph.D., professor, Ochanomizu University, Tokyo, Japan.
Correspondence concerning this article should be addressed to Ai Hayakawa, Ochanomizu University 2-1-1Otsuka, Bunkyo-ku Tokyo 112-8610, JAPAN and Saneyasu Yamaguchi, Kogakuin University 1-24-2, Nishishinjuku, Shinjuku-ku Tokyo 163-8677, Japan.

This paper focuses on the ACK packet backlog problem with the upstream TCP sessions, and proposes a CUBIC based CWND control mechanism as part of the middleware that can be implemented in the Android terminals. It utilizes measured RTTs as an indication for the TCP ACK backlog condition at the WLAN AP, and controls the upper and lower bounds of its CWND size to suppress excessive transmissions of own TCP DATA packets. The both bounds are determined by the BDP (Bandwidth-Delay Product), the number of terminals, and existing CSMA/CA performance studies. An experimental study with up to 10 Android terminals shows that the proposed mechanism can improve both aggregate throughput and fairness of the WLAN, and that it is highly effective particularly for cases where very long RTTs are observed.

Backgrounds

Related Work

TCP originally assumed that a packet drop is an indication of network congestion because the primary reason for a packet to be dropped is queuing overflow at one of the routers on the way to the other communication peer. However, wireless communications introduce other causes for packet drops such as fading, collisions, and interference. These additional causes confuse the TCP CWND control algorithm, leading to a degraded performance. Such effects of wireless communications on the TCP performance and techniques to combat those effects have been extensively studied (Prasun, Thyagarajan, Narayanan, Ragupathy, & Vaduvur, 2002; Claudio, Mario, Saverio, Sanadidi, & Wang, 2002; Luigi, Grieco, & Saverio, 2004; Shao, Tamer, & Srikant, 2006).

In the previous work (Hiromi, Saneyasu, & Masato, 2013; Ai, Saneyasu, & Masato, 2015), methods for improving WLAN throughput according to the number of terminals were proposed. In these works, WLAN terminals attempt to estimate the number of neighboring terminals that operate on the same channel by monitoring broadcast activities, and adjust its CWND size accordingly to gain its fair share. Both of these works were based on manual optimization, thus these methods require tuned table for setting CWND before applying them. Unlike these existing methods, the proposed method in this paper does not require performance tuning. In addition, the performance obtained by our method is much higher than those of these existing methods.

Android OS

This study focuses on the implementation of our proposed mechanism as a middleware of the Android platform. Android is a platform for mobile terminals whose development is led by Google, and is distributed as a package that includes an Operating System and basic applications. The source code of Android is available via the Android Open Source Project by the Open Handset Alliance (Android Open Source Project).

In this work, Android 4.1 (JellyBean) was used as the baseline, because it is a relatively new version released in July 2012 and currently has the highest share (44.5% as of 2 February, 2015) among all Android versions. Android is built based on the Linux kernel, which provides basic capabilities such as multi-stack networking, multitasking, virtual-memory management, and virtual machines. Therefore, this work can be applied to other Linux-based terminals or systems, although the performance evaluation was conducted with Android terminals in this paper.

Since the default TCP version in Linux is CUBIC, Android adopts CUBIC as the transport protocol. CUBIC, like any other transport protocols, controls the rate of DATA packet transmissions based on CWND, the maximum number of packets that can be transmitted without receiving an ACK packet from the DATA

packet receiver. Setting an appropriate CWND is the key to achieving high throughput, which is the primary difference between various versions of TCP. In CUBIC, CWND is increased gradually per receiving an ACK and halved every time a packet loss is experienced as shown in Figure 1. As the CWND size is reduced upon a packet loss, it is called a loss-based TCP. Other CWND control mechanisms used in TCP Vegas and TCP Westwood are based on the observed RTTs, and are called a delay-based TCP (Saverio, Claudio, Mario, Sanadidi, & Wang, 2001).

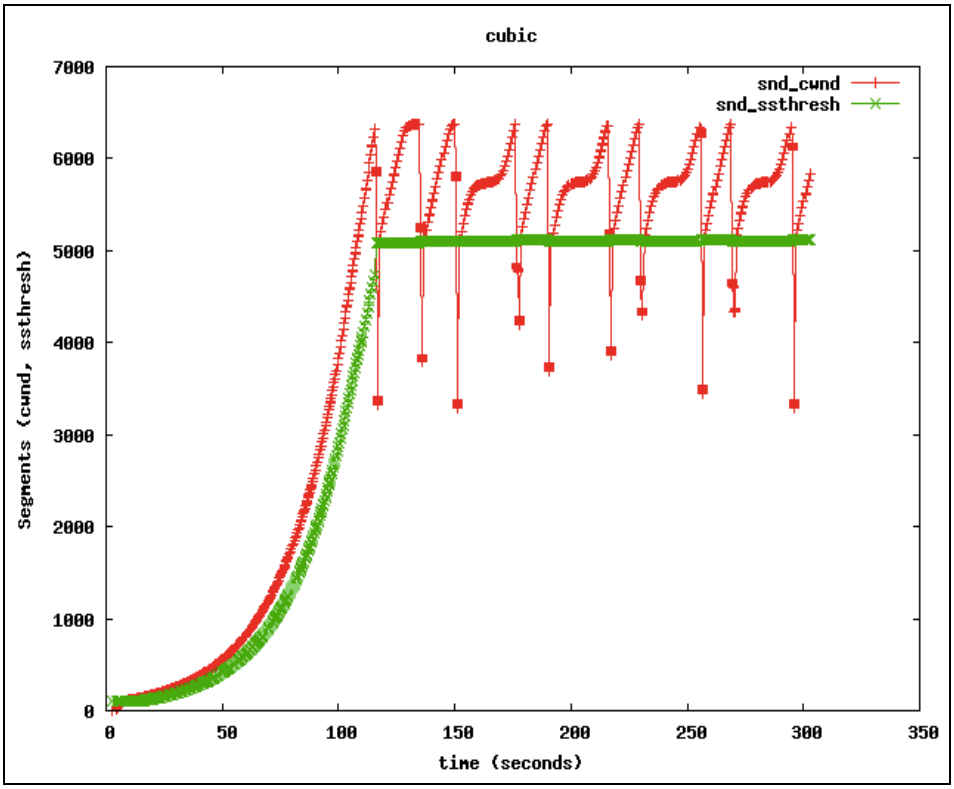

Figure 1. Behavior of CUBIC TCP.

CWND-Controlling Middleware

This subsection describes the CWND-controlling middleware that has been implemented in our previous work (Hiromi, Saneyasu, & Masato, 2013). This middleware controls CWND if the terminal originates the TCP traffic and is connected to the server via a WLAN to address congestion among the AP and other Android terminals.

This middleware can be divided into two parts. One part adjusts the congestion control, using the process interface to prevent segments from overflowing and filling the bandwidth. The notification is broadcasted by the UDP every 0.3 seconds from the other part because the kernel parameters frequently change. The adjustment is executed every 10 seconds because the number of mobile hosts changes less often, and this lower frequency is sufficient for collecting information from all hosts when considering the notification interval and the arrival rate of the notifications.

This system adjusts the congestion control algorithm based on the status of the other hosts that are connected to the identical AP to organize all of the traffic. To improve the traffic control quality, the value of the CWND, which denotes the number of segments on the fly, should occupy the entire bandwidth without

exceeding it. However, the detailed communication situation of individual terminals could not be understood in the previous study because each terminal was controlled based only on the information about the number of neighboring terminals and their CWND. In addition, another problem was noted in which we had to re-determine the value to adjust the CWND when the system environment was changed because the value was fixed in the previous study. Therefore, we solve these problems in this study and aim to improve the performance of this system and further enhance the convenience.

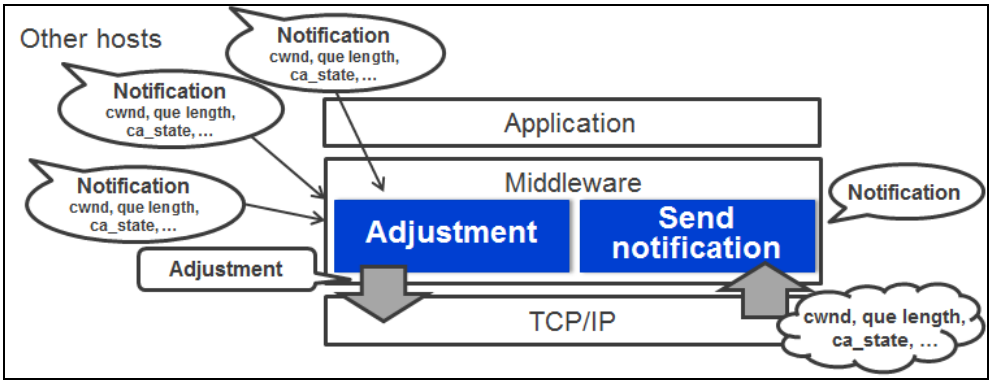

Figure 2. Summary of the system.

Kernel Monitoring Tool

Our previous work (Kaori, Saneyasu, & Masato, 2011) successfully embedded a system tool called the Kernel Monitoring Tool in the Android platform to analyze the connection status of a mobile host. This tool allows users to monitor parameters in the kernel processing at the mobile host including the CWND, CA state, and socket buffer queue. These parameters are defined in the TCP implementation of the Linux Kernel code, and applications in the user space can generally never observe or even recognize them. Using the Kernel Monitoring Tool, our middleware can access the current values of these parameters in the kernel memory space.

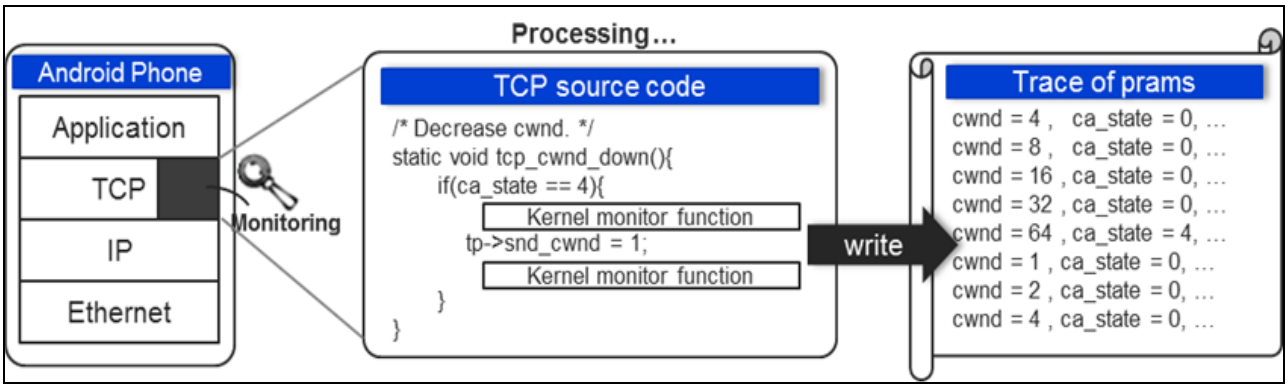

Figure 3. Kernel monitoring tool.

Proposed Mechanism

Overview of Our Middleware

In this section, we summarize the CWND-controlling middleware used in this study. Our previous middleware introduced in previous section consists of dispatch components and reception components. In the

proposed mechanism, we unified both components, and as a result, both dispatch and reception functions make use of information from the kernel monitoring tool, which can be ideally tuned.

Figure 4 shows the composition of the middleware after the modification. The values of RTT and minimum RTT (min rtt) are acquired by observing data from the kernel monitoring tool constantly. A value of min rtt is updated by overwriting with the smallest RTT during the communication. Based on the acquired values, RTT Ratio (ratio rtt) is obtained with Expression (1), which indicates increase and decrease of RTT.

$$ratio\_rtt = \frac{RTT}{\min\_rtt} \tag{1}$$

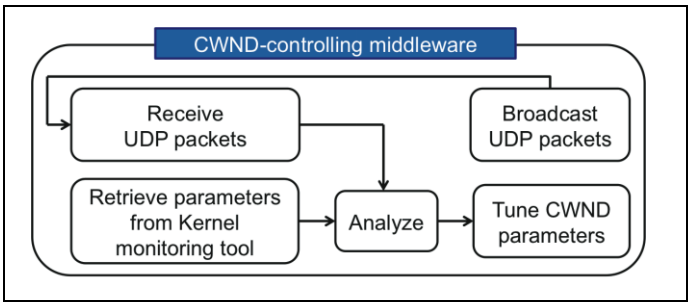

Figure 4. Composition of the middleware after modification.

Simultaneously, traffic condition is predicted by receiving packets and knowing the communication situation of other terminals. This system sets levels of upper and lower limit for the CWND based on the RTT ratio and the number of communication terminals using the process interface, which is controllable from an outside process of kernel and can be tuned up for optimization. Using this method, the control system can limit the quantity of traffic outbreak and share a bandwidth of a terminal, after the initial communication is enabled by setting a CWND level at an appropriate value.

The communication situation is checked approximately every 0.5 seconds in the current implementation. CWND level is modified when the middleware detects another terminal that shares the same AP begins communication and the RTT values suddenly increase as a result. With 0.5 seconds frequency, the kernel monitoring tool can grasp the timely situation moderately and optimize it. If the frequency is too high, it may become an overhead.

In this middleware, we do not modify the congestion control algorithm itself of the basic TCP, which functions similar to the default case and should be good for the interoperability. Nevertheless, the communication is optimized by setting the levels of upper and lower limit for the CWND, and the congestion control is adjusted based on the communication situation of AP surroundings.

Expressions for Proposed Control Mechanism

In each delay environment, we decide the levels of upper and lower limit for the CWND referring to Expressions (2), (3), and (4). In these expressions, BWmax is the maximum throughput obtained in the environment. And f(x) in expression (4) is the monotonically decreasing function for adjusting negative effect of increases of the number of devices on performance. It is widely recognized that the obtained throughput is decreased as the number of attending devices increases. In work (Makiko & Masato, 2012), the practical relation between the obtained performance and the number of devices are presented, and it is approximately described as shown in expression (4). The limit values are decided based on these expressions.

The calculated values are maximum CWND value. These values are transmitted to kernel every 0.5 second by writing to/process interface. The current RTT (RTT) and the minimum RTT (min rtt) are obtained every 0.5 second by reading kernel monitor files in/process interface. min rtt is the smallest RTT through the connection.

The raito rtt is determined by RTT and min rtt using the expression (5).

$$\text{Idealcwnd} = \frac{\text{Bandwidth[Mbps]} \times \text{RTT[sec]}}{\text{Segmentsize(1.5Kbyte)} \times \text{number of terminals}} \tag{2}$$

$$\text{Bandwidth[Mbps]} = \text{BWmax[Mbps]} \times \text{f (number of terminals)} \tag{3}$$

$$\text{f (number of terminals)} = \text{number of terminals}^{-0.15} \tag{4}$$

$$ratio\_rtt = \frac{RTT[\text{sec}]}{\min\_rtt} \tag{5}$$

When the system starts, the proposed control using expression (2) is disabled. If ratio rtt is greater than 6.0, the system assumes that the network is congested and the control is enabled. If ratio rtt is greater than 5.0 with the control enabled, our system sets CWND to 1.

Performance Evaluation

Basic Scalability Test

The experimental system in Figure 5 was used for the performance evaluation. Android phones 1-10 were connected to an AP over 802.11g, and the AP was connected to the server host using the wired route. To emulate artificial delay and loss rate, a network emulator, Dummynet, was inserted between the AP and the server host. In this environment, wired parts are connected with higher rate because of Gigabit Ethernet, whereas a bandwidth is only about 20 Mbps in wireless parts. Thus, the radio transmission sections between an AP and terminals should become a bottleneck. Especially, when the number of the terminals connected at the same time increases, AP may cause a buffer overflow and the length of a packet cue for transmission should increase. As a network benchmark tool, Iperf for Android was installed on all Android phones. The specifications of each device used in the experiment are shown in Table 1. BWmax is 19 Mbps in this environment.

Figure 6 shows the relationship between the number of terminals and the throughput. The blue line shows the average throughput, and the red line shows the total throughput. The total performance decreases when the number of terminals increases in each delay environment.

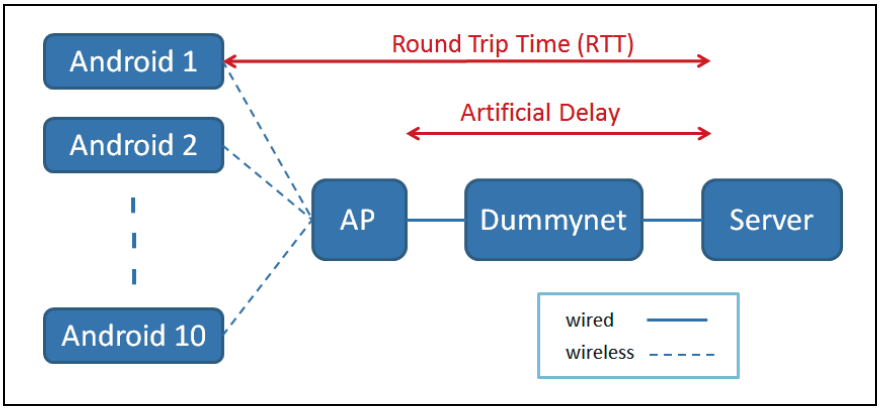

Figure 5. Experimental topology.

Table 1

Specifications of Devices

Android	Model number	Nexus S
	Firmware version	4.1.1
	Baseband version	I9023XXKD1
	Kernel version	3.0.31-ai
	Build number	JRO03L
Server	OS	Ubuntu 12.04 (64bit) / Linux 3.0.1
	CPU	Intel(R) Core 2Quad CPU Q8400
	Main memory	7.8 GiB
AP	Model	MZK-MF300N(planet)
	Sommunication system	IEEE 802.11g

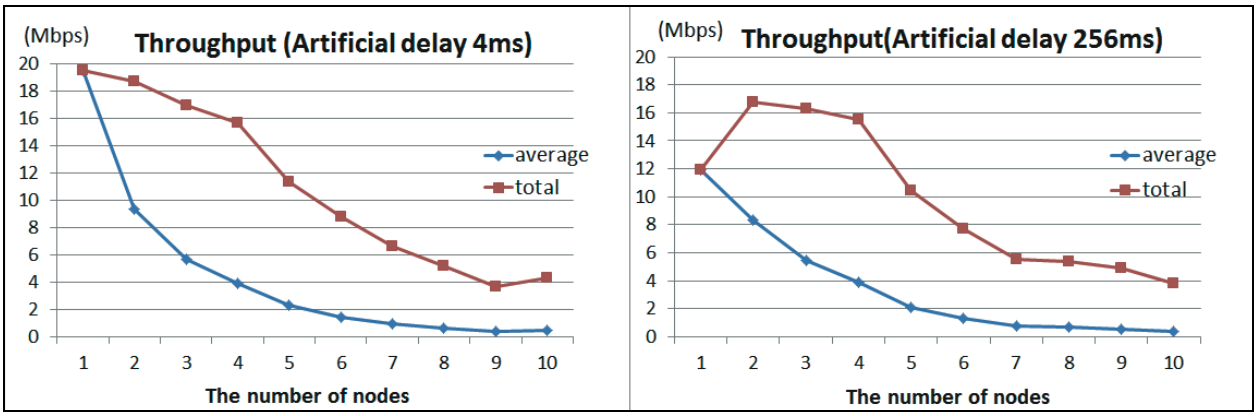

Figure 6. Relations between the number of terminals and the throughput.

Next, we measured the CWND using the kernel monitoring tool and the end-to-end RTT using the ping command to determine the cause of performance decline. Figures 7 and 8 show CWND and RTT when 1, 5, or 10 terminals communicate in each delay environment.

It was found that although CWND behaved as CUBIC TCP when the number of terminals was small, congestion control was disturbed when the number of terminals increased. In addition, the graphs of RTT, which is end-to-end delay time, showed that it significantly increased in the case of ten terminals. In both cases, more than 12 seconds RTT was detected, which is extremely larger than delay time introduced by the network emulator (4 ms or 256 ms). Thus, the large RTT was observed in many communication cases. Moreover, not a particular packet was highly delayed accidentally, but instead, a series of packets was highly delayed for bursting. They were accumulated in a buffer, which remained in a wait condition.

Moreover, after packets were captured in the server side and the data were analyzed with Wireshark, it was confirmed that duplicate ACK was transmitted from the server side, whereas packets were sent normally from the terminal side. A time transition of CA STATE that indicates the state of TCP acquired by the kernel monitoring tool is shown in Figure 9. In these states, “0” means “Open”: normal state, “1” means “Disorder”: replacement of packets, “2” means “CWR”: congestion notice, “3” means “Recovery”: fast retransmission, and ”4” means ”Loss”: timeout and loss of packets. Many cases in the Fast Retransmission state indicated that state 3 was detected because buffer overflow occurred in AP, then RTT became longer, and duplicate ACK was returned when the number of communication terminals was large.

From these results, it is confirmed that the communication performance decreases as RTT increases larger, independent of the delay time of the wired network part, when many terminals simultaneously communicate. Therefore, we add the RTT and its minimum value as parameters used for the kernel monitoring tool. Here, the RTT is a value measured in the TCP implementation of the Android OS, and this is a real value for packets coming and going over the delay line. On the other hand, the minimum RTT is a value measured in the state with a network of small load, and this almost equals the artificial delay time set in the Dummynet. Congestion among the current traffic can be grasped by observing these quantities.

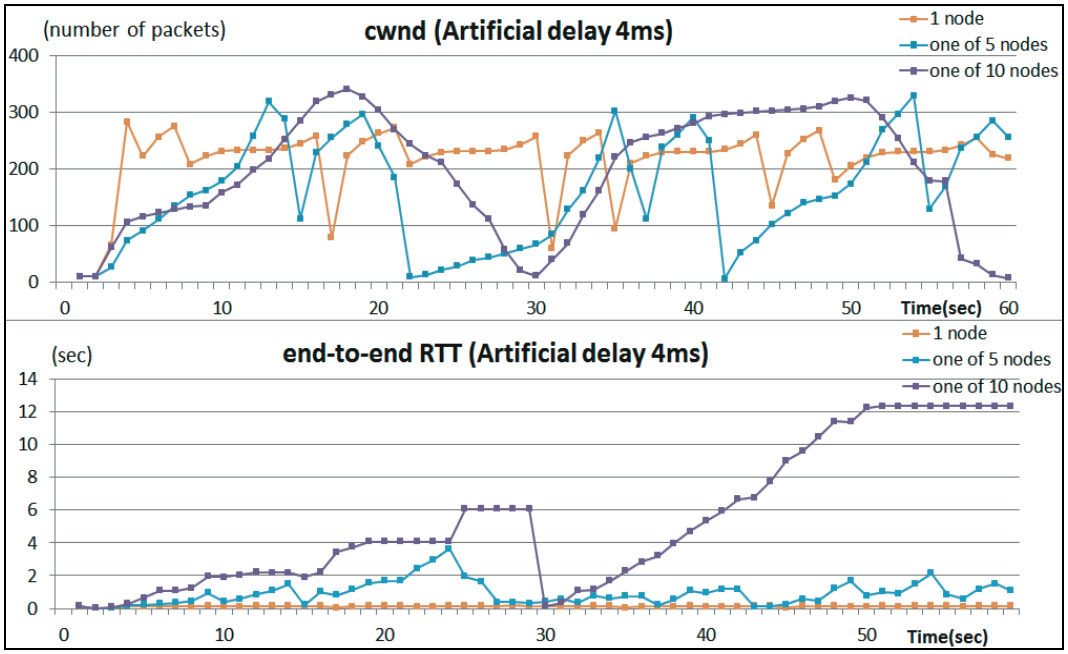

Figure 7. Transition of the CWND and RTT for an artificial delay of 4 ms.

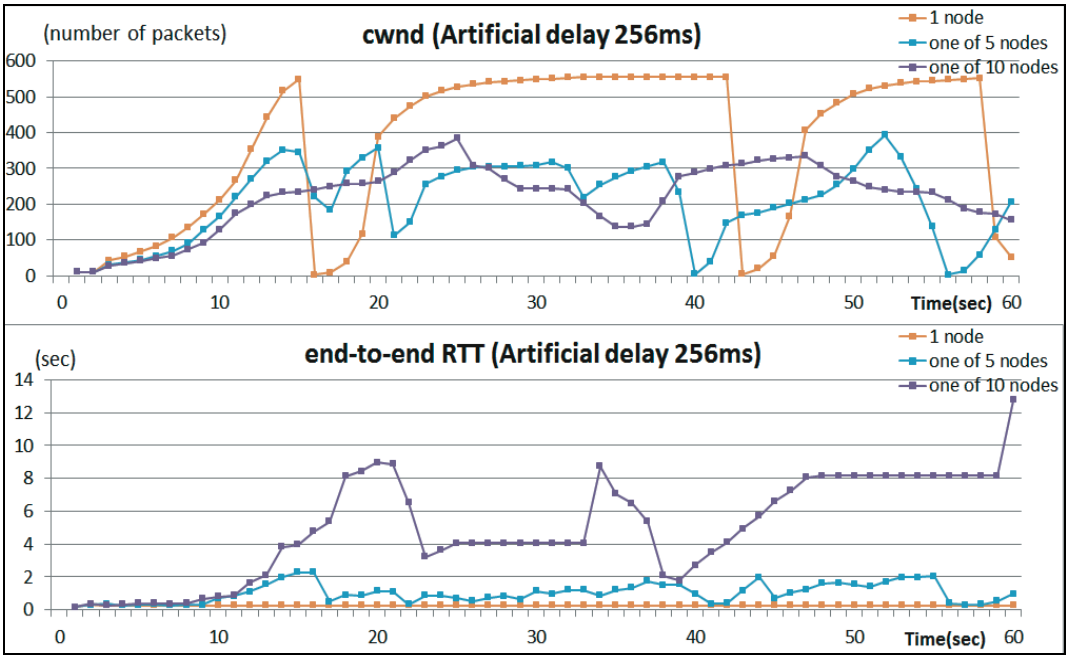

Figure 8. Transition of the CWND and RTT for an artificial delay of 256 ms.

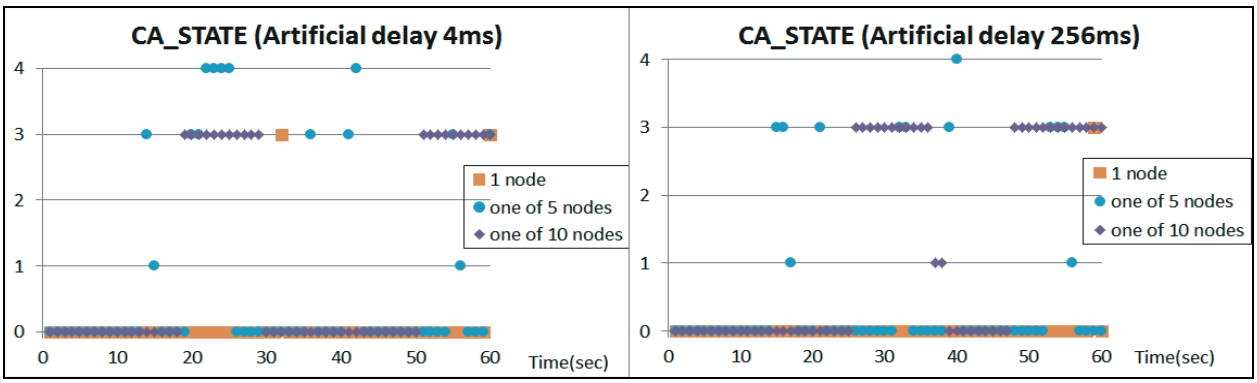

Figure 9. The state of TCP.

Correlation Between Throughput and RTT

We investigated the relationship between the RTT and the communication performance using an Android terminal with the kernel monitoring tool after the modification to obtain an RTT value in the experimental environment. The procedure is identical to that used in previous experiment to clarify the effect of an increase in RTT on the communication performance.

The time transition of the RTT and the throughput, which were obtained using the kernel monitoring tool, for the communication of 10 Android terminals with an artificial delay of 256 ms are shown in Figure 10. This graph shows that the RTT is small when the throughput is high and that the RTT is large when the throughput is low. According to this result, a large increase in RTT significantly reduces the communication performance, as expected. Therefore, the communication control that considers the increase and decrease in RTT can effectively improve the transmission speed.

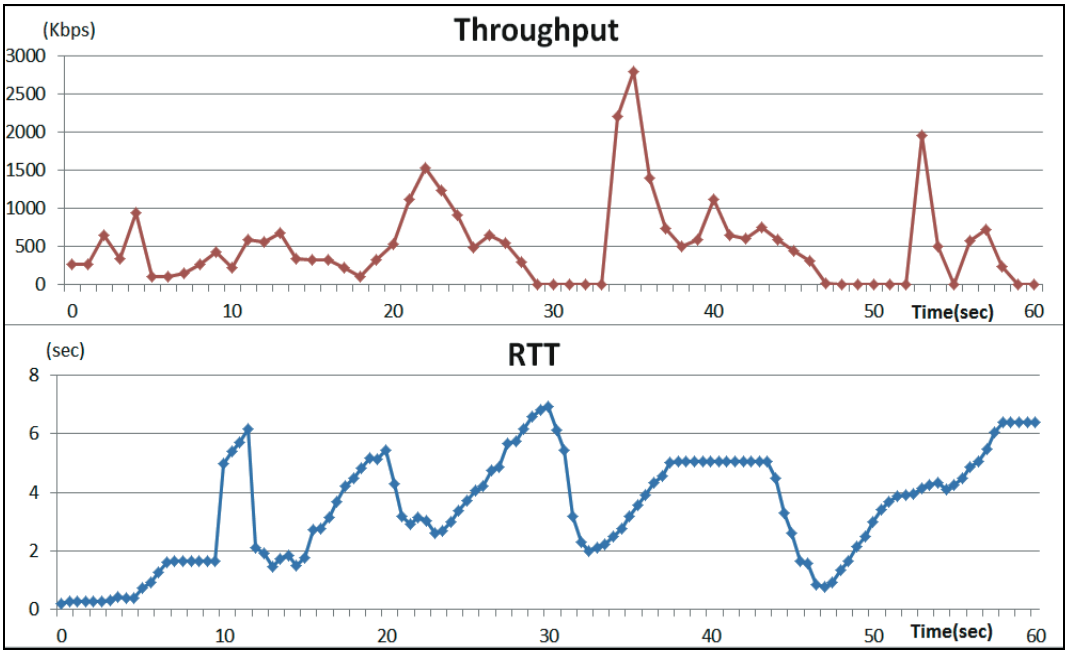

Figure 10. Time transition of the RTT and the throughput.

Case of Proposed Mechanism in All Terminals

This subsection shows the effects of the proposed mechanism when all terminals utilize our middleware.

The results of the total throughput are shown in Figure 11. The blue graph shows the throughput of the default condition without the middleware, and the red graph shows that of the adjusted condition with our middleware. The performance was significantly improved by introducing the middleware, regardless of the artificial delay time. Particularly, in the case of 10 nodes, the performance improved by 3.35 times.

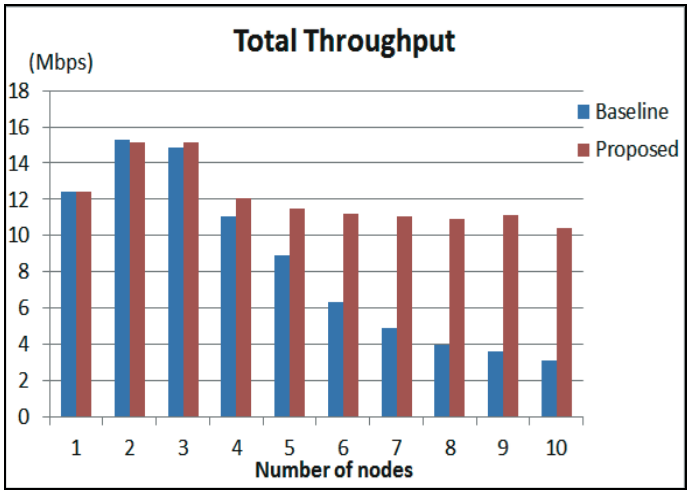

Figure 11. Total throughput.

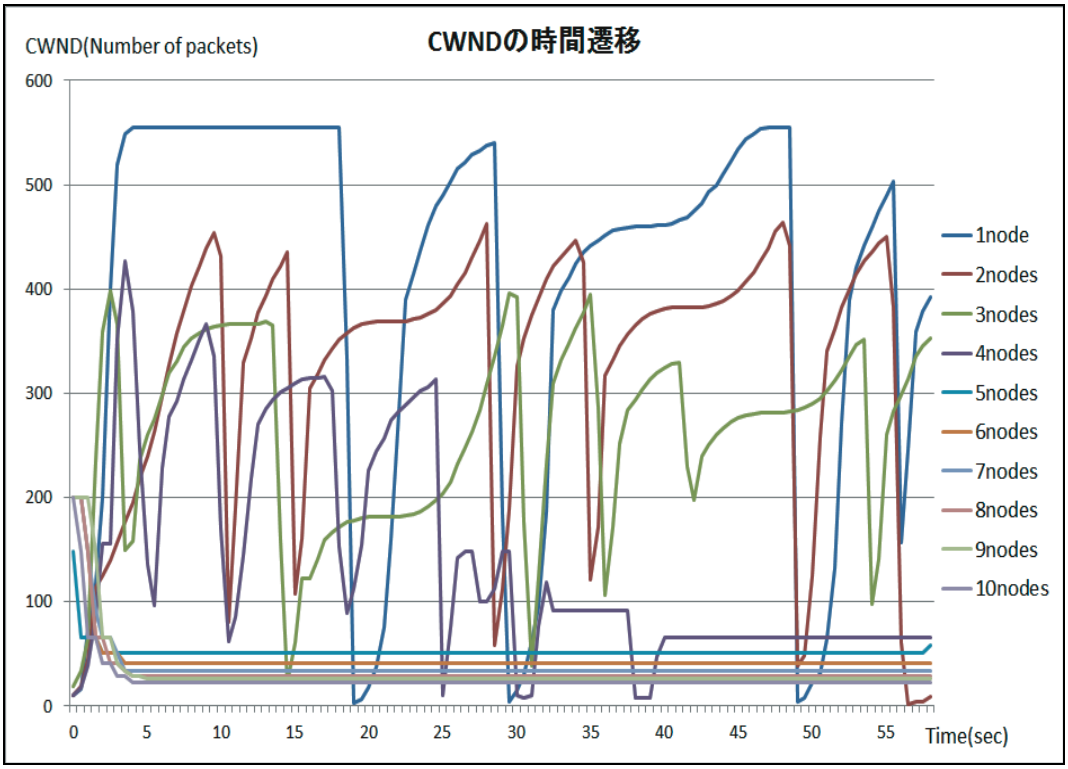

Figure 12. Behavior of CWND.

An example of behavior of the adjusted CWND is shown in Figure 12. In the cases of one, two, and three node(s), the network was not congested and ratio rtt did not achieve 6.0, and then the control was not enabled. In the case of four nodes, ratio rtt achieved 6.0 and the control was enabled, and then CWND was adjusted.

Figure 13 depicts transition of ratio rtt. The figure shows that the system does not allow ratio rtt to keep larger than 6.0. In the case of four nodes, ratio rtt achieved 6.0 and it was decreased immediately. In the cases of five or more nodes, ratio rtts were controlled to small values. These figures indicate that our proposed mechanism has controlled congestion suitably.

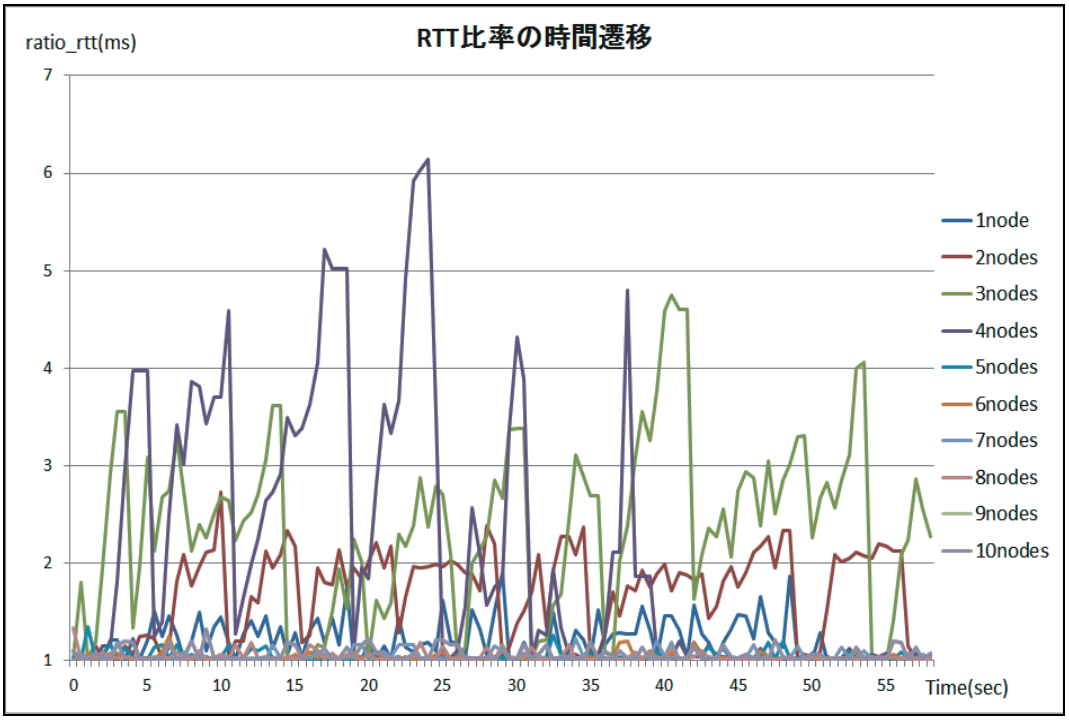

Figure 13. Behavior of ratio rtt.

Next, we evaluate the communication performance using the Fairness Index (Chiu & Jain, 1989) to confirm that the available bandwidth is fairly assigned to each terminal. The Farness Index indicates the equitableness, and a calculated value of 1 means the highest fairness. Expression (6) shows the calculation method.

$$\text{FairnessIndex : fi} = \frac{(\sum_{i=1}^{k} x_i)^2}{k\sum_{i=1}^{k} x_i^2} (1 \le i \le k) \tag{6}$$

An evaluation of the fairness is shown in Figure 14. The blue bars show fairness of the baseline, and the red ones show that of the proposed mechanism. The Fairness Index of the proposed mechanism always keeps approximately 1, whereas the fairness is spoiled when there is no middleware and the number of terminals increases. This result demonstrates that the fairness is also remarkably improved by introducing our middleware. According to the above results, we can conclude that our mechanism successfully improves the transmission speed when many terminals communicate. The speed increases by more than 3 times by introducing the suggested middleware. In addition, the fairness can be generally improved with the middleware. Comparison between performance of the proposed mechanism in this paper and that of the existing works (Ai, Saneyasu, & Masato, 2015) is presented in Appendix. Significant performance improvement is observed when the number of nodes is large.

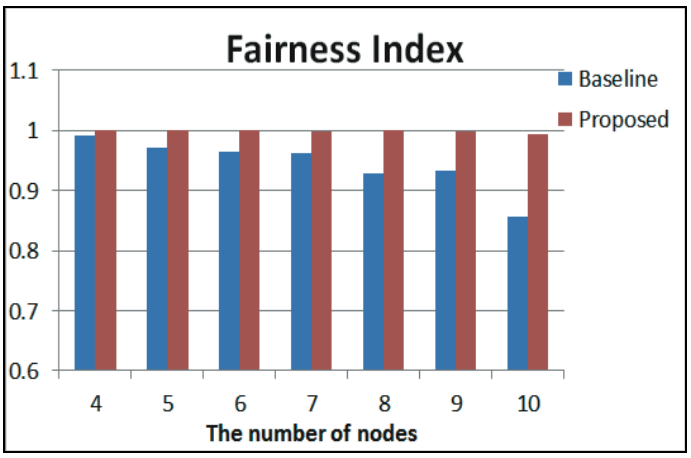

Figure 14. Fairness evaluation.

Case of Coexistence

In this subsection, the number of terminals that utilize our middleware is varied, and the rest of the terminals are leaved not to utilize our middleware. The performance of WLAN in such situations is important because it is unrealistic to force all users to install our middleware in their terminals, and therefore the deployment of our proposed mechanism is expected to be only partial in many cases. In the experimental setup, the artificial delay is set to be 256 ms, the same with the settings in previous experiment, and the total number of terminals in WLAN is varied to 4, 7, and 10. Figures 15 through 17 show the throughput performances for the 4, 7, and 10 terminals cases, respectively. In these figures, the blue line shows the total throughput, the red bars indicate the average throughput of each terminal that utilizes our middleware, and the green ones show the average throughput of the terminals without middleware. The average throughput of each terminal without middleware tends to be a little higher in comparison with a terminal with middleware, because the terminal that does not have middleware tends to communicate aggressively. However, the total throughput with middleware terminals is higher than the case that no terminal has middleware, although the terminals with middleware and without middleware are mixed in this environment. Thus, it is possible to improve overall communication performance by adjusting only some terminals with our middleware.

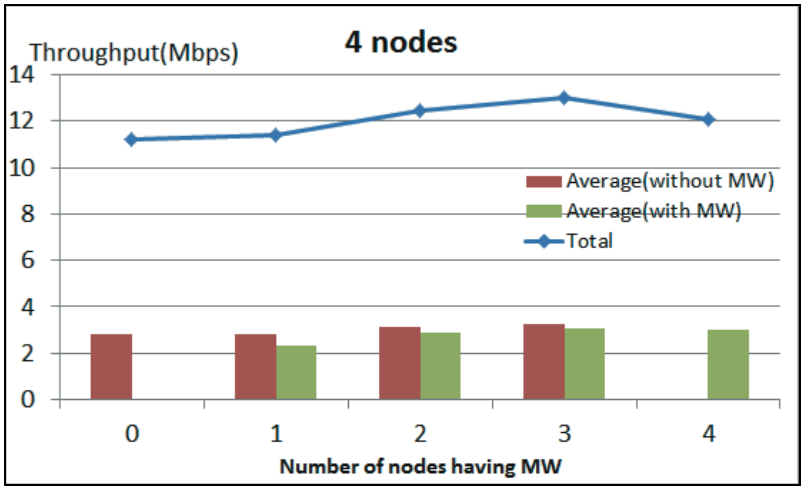

Figure 15. Throughput when 4 terminals communicate.

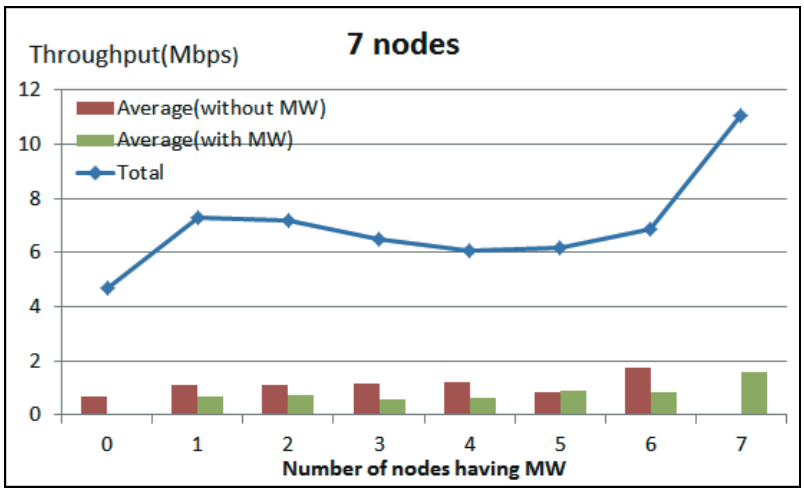

Figure 16. Throughput when 7 terminals communicate.

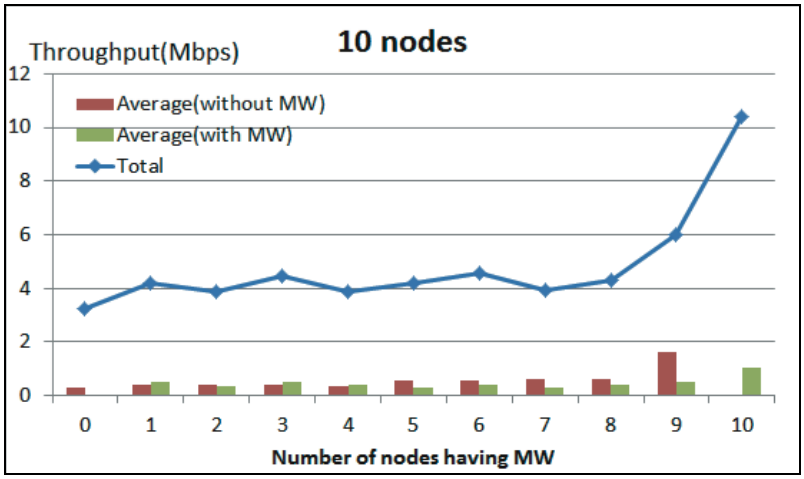

Figure 17. Throughput when 10 terminals communicate.

Discussions

In previous section, we evaluated performance in the environment where terminals that did not have our middleware were mixed. However, there is also a possibility that not only Android terminals but also notebook PCs and other terminals share the same AP in a real environment. Therefore, we also evaluated in an environment where Windows and Mac OS PCs were mixed with Android terminals having our middleware, and succeeded in improvement of the overall transmission rate.

In the previous method, the CWND was adjusted only by the number of terminals connected to the same AP. In our previous study, RTT was added to parameters for adjusting CWND. Unfortunately, the method in that requires parameter tuning before applying it and obtained performance strongly relies on parameter tables. Throughput is increased compared with these methods. Especially, when the number of terminals is large, significant performance improvement is achieved.

In our evaluations, all terminals access the same server but the connected server of each terminal is different in a real environment. In other words, the delay line in the wired part is different by each terminal, even if the wireless part is shared. Therefore, we executed the following experiment; different pipes those were

assigned to each terminal were established on Dummynet, and delay time was fluctuated differently depending on the terminals. As a result, even though the delay time of Dummynet fluctuated, it almost did not change the end-to-end RTT. Thus, the bottleneck exists in the wireless part especially when many terminals share the same AP. Of course, it is possible the RTT of wired part affects the total performance if some kinds of troubles happen on a wired connection. However, such cases are discussed in other literatures and out of scope for our study.

Conclusions

This study has focused on the ACK packet backlog problem with the upstream TCP sessions, and has proposed a CUBIC based CWND control mechanism that utilizes the RTT as an indication for the TCP ACK backlog condition at the WLAN AP, and controls the upper and lower bounds of its CWND size to suppress excessive transmissions of own TCP DATA packets. Unlike the existing works, the proposed mechanism is based on CSMA/CA performance study and does not require parameter tuning.

The experimental results with up to 10 Android terminals show that the proposed mechanism can improve both aggregate throughput and fairness of the WLAN. In particular, three times TCP throughput improvement is observed when many terminals are associated with the AP.

There are a few possible directions that can extend this study. We plan to evaluate our mechanism in situations where the numbers of active terminals and TCP sessions change dynamically. Further, the WLAN AP used in this study supports only data rates defined by the IEEE 802.11b/g, but more recent APs that offer wider ranges of data rates such as IEEE 802.11n or 802.11ac should be used in the evaluation of the proposed CWND controlling middleware.

References

Ai, H., Saneyasu, Y., & Masato, O. (2015). Reducing the TCP ACK packet backlog at the WLAN. AccessPoint, *In Proc. the 9th ACM International Conference on Ubiquitous Information Management and Communication (IMCOM2015)*, 5-4, January 2015.

Android Open Source Project. http://source.android.com

Cuina, Z., Jing, H., & Lianfeng, S. (2010). A CSMA/CA MAC protocol of cognitive networks based on IEEE 802.11, *White paper*, National Mobile Communications Research Laboratory, Southeast University.

Claudio, C., Mario, G., Saverio, M., Sanadidi, M. Y., & Wang, R. (2002). TCP westwood: End-to-end congestion control for wired/wireless networks. *Wireless Networks Archive*, *8*(5), 467-479.

Dimitrios, K., Jagadeesh, D., Prashant, G., Sonia, F., & Charlie Hu, Y. (2007). On TCP throughput and window size in a multihop wireless network testbed. *In Proc. WiNTECH'07*, 51-58.

Dina, K., Mark, H., & Charlie, R. (2002). Congestion control for high bandwidth-delay product networks. *In Proc. SIGCOMM'02*, 89-102.

Chiu, D-M., & Jain, R. (1989). Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems, 17*, 1-14.

Hiromi, H., Saneyasu, Y., & Masato, O. (2013). A proposal on cooperative transmission control middleware on a smartphone in a WLAN environment. *In Proc. the 9th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob2013)*, 710-717.

Kaori, M., Saneyasu, Y., & Masato, O. (2011). Kernel monitor of transport layer developed for android working on mobile phone terminals, *Proceedings of The Tenth International Conference on Networks (ICN)*, pp. 297-302.

Luigi, A., Grieco, & Saverio, M. (2004). Performance evaluation and comparison of westwood+, New Reno, and Vegas TCP congestion control. *ACM SIGCOMM Computer Communications Review*, *34*(2), 25-38.

Xu, L., Harfoush, K., & Rhee, I. (2003). *Binary increase congestion control for fast, long distance networks, proceedings of Tech. report*, Computer Science Department, NC State University.

Makiko, M., & Masato, O. (2012). Multiple access in MAC layer based on surrounding conditions of wireless stations. *In Proc. the 1st International Workshop on Vehicular Communications and Applications (VCA2012) in Conjunction with the 11th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net2012)*, 133-140.

Naok, S., Hiroyuki, T., Yoshiaki, O., Shinichi, A., & Naoki, S. (2007). Middleware for controlling multiple bearer utilization using communication policy, IPSJ SIG technical reports. MBL, ISP SIG mobile computing and ubiquitous communication, C09196072. *Information Technology Standards Commission of Japan*, *44*, 7-12.

Prasun, S., Thyagarajan, N., Narayanan, V., Ragupathy, S., & Vaduvur, B. (2002). WTCP: are liable transport protocol for wireless wide-area networks. *Wireless Networks-Selected Papers From Mobicom'99 Archive, 8*(2/3), 301-316.

Saverio, M., Claudio, C., Mario, G., Sanadidi, M. Y., & Wang, R. (2001). TCP westwood: bandwidth estimation for enhanced transport over wireless links. *In Proc ACM SIGMOBILE* 7/01, Rome, Italy.

Ha, S., Rhee, I., & Xu, L. (2008). CUBIC: a new TCP-friendly high-speed TCP variant, ACM SIGOPS operating systems review. *Research and Developments in the Linux Kernel, 42,* 64-74.

Shao, L., Tamer, B. & Srikant, R. (2006). TCP-Illinois: a loss and delay-based congestion control algorithm for high-speed networks. *Valuetools '06 Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, 55.

Vasilios, A., Siris, & George Stamatakis. (2006). Optimal CWmin selection for achieving proportional fairness in multi-rate 802.11e WLANs: test-bed implementation and evaluation. *In Proc. WiNTECH'06*, 41-48.

Yuki, T., & Hitoshi, A. (2011). Study of a multi-path communication protocol in wireless environment, IPSJ SIG technical reports. *EIP Electoronic Intellectual Property, 09196072, Information Technology Standards Commission of Japan*, 11, 1-7.