Reducing the TCP ACK Packet Backlog at the WLAN Access Point

Ai Hayakawa
Ochanomizu University
2-1-1, Otsuka, Bunkyo-ku
Tokyo 112-8610, JAPAN
ai@ogl.is.ocha.ac.jp

Saneyasu Yamaguchi
Kogakuin University
1-24-2, Nishishinjuku, Shinjuku-ku
Tokyo 163-8677, JAPAN
sane@cc.kogakuin.ac.jp

Masato Oguchi
Ochanomizu University
2-1-1, Otsuka, Bunkyo-ku
Tokyo 112-8610, JAPAN
oguchi@computer.org

ABSTRACT

Modern loss based Transmission Control Protocols take aggressive congestion window (CWND) control strategies in order to gain better throughput, but 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. 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.

Categories and Subject Descriptors

C.2.5 [Computer Systems Organization]: COMPUTER-COMMUNICATION NETWORKS[Local and Wired-Area Networks]; D.1.1 [Software]: PROGRAMMING TECHNIQUES—Applicative(Functional) Programming

General Terms

Measurement

Keywords

TCP, Congestion control, RTT, Android

IMCOM(ICUIMC) '15 January 8-10, 2015, Bali, Indonesia.

1. INTRODUCTION

Modern loss based TCPs such as BIC [1] and CUBIC [2] take aggressive CWND control strategies in order 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, since a wireless link can usually offer much narrower bandwidth 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 [3] based 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 that can be implemented in the Android terminals. It 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. 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.

The rest of the paper is organized as follows. Section 2 describes some related works. Section 3 describes the backgrounds of this work, including our previous studies. Section 4 propose a new mechanism. Section 5 evaluates our proposal and discussions the results and problems. Finally, Section 6 presents our conclusions and future works.

2. RELATED WORK

TCP has originally assumed that a packet drop is an indication of network congestion, since the primary reason for a packet to be dropped is the queueing overflow at one of the routers along the path to the other communication peer. However, wireless communications introduce other causes for packet drops such as fading, collisions and interference, which confuse the TCP CWND control algorithm to lead to suboptimal performance. Such effects of wireless communications on the TCP performance as well as techniques to combat those have been extensively studied [4][5][6][7] in the literature.

Our previous work [8] is one of them, in which each WLAN terminal attempts to estimate the number of neighboring terminals that operate on the same channel by monitoring

broadcast activities, and adjusts its CWND size accordingly to gain its fair share. Details of this work are described in Section 3.2. This paper proposes an additional CWND size control mechanism to the previous work by estimating the TCP ACK packet backlog condition by means of TCP RTT in order to further improve the TCP performance.

3. BACKGROUNDS

3.1 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 [9].

In this work, Android 4.1 (JellyBean) has been used as the baseline since it is a relatively new version released in July 2012, and currently has the highest share (27.8% as of 7 July, 2014 [10]) among all the Android versions. Please note that Android is built based on the Linux kernel, which provides basic capabilities such as multistack networking, multitasking, virtual-memory management and virtual machines. Therefore, this work can be applied to any other Linux based terminals or systems although the performance evaluation has been conducted only with Android terminals.

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 [11].

3.2 CWND-controlling middleware

This subsection describes the CWND-controlling middleware that has been implemented in our previous work [8]. This middleware controls CWND if the terminal originates TCP traffic and is connected to the server via a WLAN, in order to address congestion among the AP and other Android terminals.

This middleware can be divided into two parts, as shown in Figure 2. One part adjusts the congestion control, using the process interface to prevent segments from overflowing and filling the bandwidth. The notification is broadcast by 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 to collect information from all hosts, considering the notification interval and the arrival rate of the notifications.

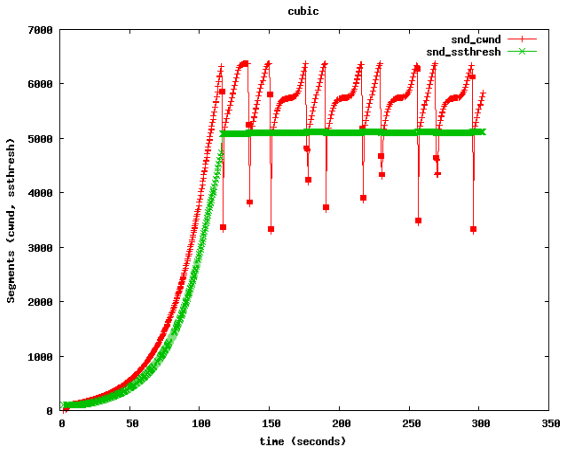

Figure 1. Behavior of TCP CUBIC

This system adjusts the congestion control algorithm based on the status of other hosts that are connected to the same AP to organize all of the traffic. To improve the traffic control quality, the value of CWND, which denotes the number of segments on the fly, should occupy the entire bandwidth without exceeding it. However, it could not support the detailed communication situation of individual terminals because each terminal in the previous study was controlled based only on the information about the number of neighboring terminals and the CWMD. Therefore, we suggest a system in which each terminal cooperates in an appropriate form and the overall communication performance is improved by monitoring the actual value of RTT at the time of communication and dynamically managing the change.

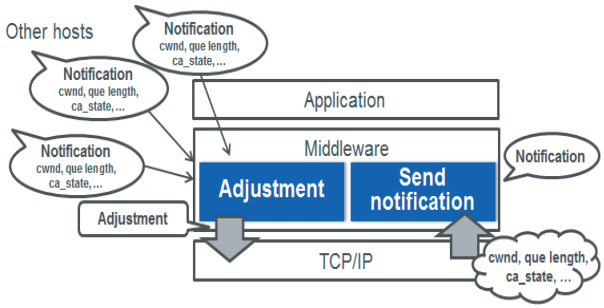

Figure 2. Summary of the system

3.3 Kernel monitoring tool

Our previous work [12] successfully embedded a system tool called Kernel Monitoring Tool in the Android platform in order to analyze the connection status of a mobile host. As shown in Figure 3, it allows users to monitor parameters in the kernel processing at the mobile host, which include CWND, CA state, and socket buffer queue. They 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. By means of Kernel Monitoring Tool, our middleware can access the current values of these parameters in the kernel memory space.

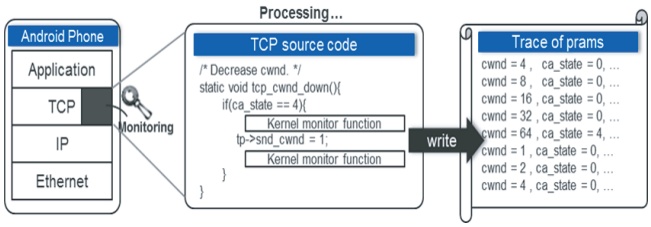

Figure 3. Kernel Monitoring Tool

4. PROPOSED MECHANISM

4.1 Overview of our middleware

In this section, we summarize the CWND-controlling middleware used in this study. Our previous middleware introduced in section 3.2 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.

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

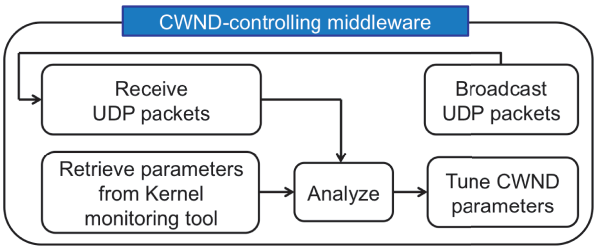

Figure 4. Composition of the middleware after modification

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 proc 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 surrondings.

4.2 Expressions for proposed control mechanism

In each delay environment, we dicide the levels of upper and lower limit for the CWND referring to Expressions 2, 3, and 4 [13]. In these expressions, the Bandwidth Delay Product (BDP) is filled with data transferred by connections that are assigned the divided bandwidth. The limit values are decided based on these expressions.

$$\begin{aligned} &Bandwidth\_DelayProduct(BDP) \\ &= Bandwidth[Mbps] \times RTT[sec] \end{aligned} \tag{2}$$

$$Idealcwnd = \frac{BDP}{Segmentsize(1.5Kbyte) \times numberofterminals} \tag{3}$$

$$IdealThroughput = \frac{CWND \times 1.5[Kbyte] \times 8}{RTT[sec]} \tag{4}$$

5. PERFORMANCE EVALUATION

5.1 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 [14], was inserted between the AP and the server host; 4 ms delay was set by assuming a low-delay environment, and 256 ms delay was set by assuming a high-delay environment.

In this environment, wired parts are connected with higher rate because of Gigabit Ethernet, whereas a bandwidth is only about 20Mbps in wireless parts. Thus, the radio transmission sections between an AP and terminals should becomes 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 [15] was installed on all Android phones. The specifications of each device used in the experiment are shown in Table 1

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	7.8GiB
AP	Model	MZK-MF300N(Planet)
	Sommunication system	IEEE 802.11g

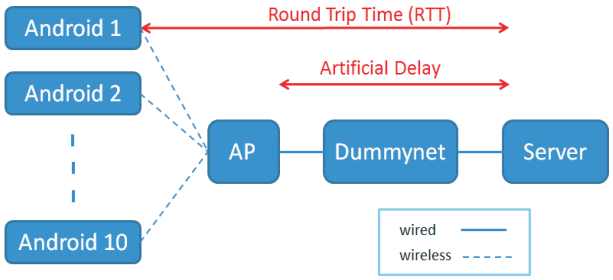

Figure 5. Experimental topology

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.

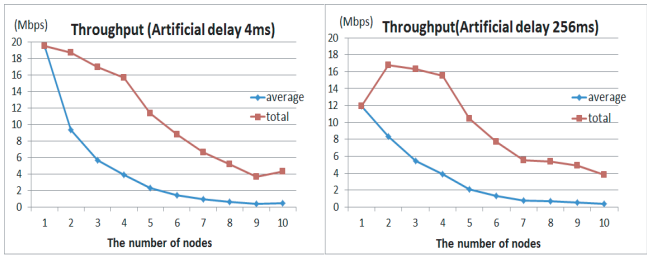

Figure 6. Relations between the number of terminals and the throughput

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 TCP CUBIC 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 were 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 was analyzed with Wireshark [16], 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.

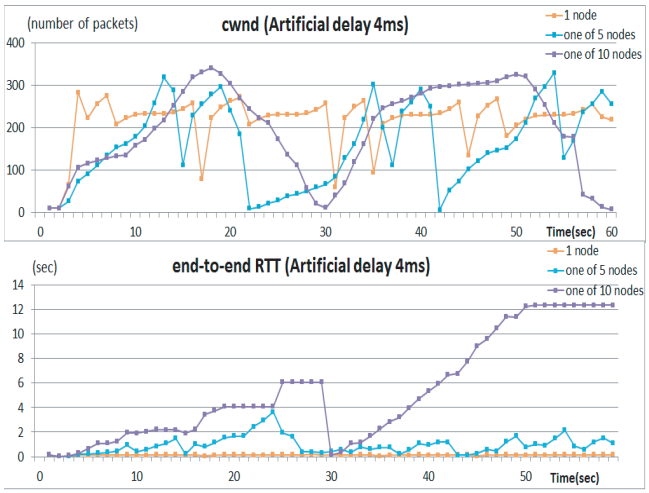

Figure 7. Transition of the CWND and RTT for an artificial delay of 4 ms

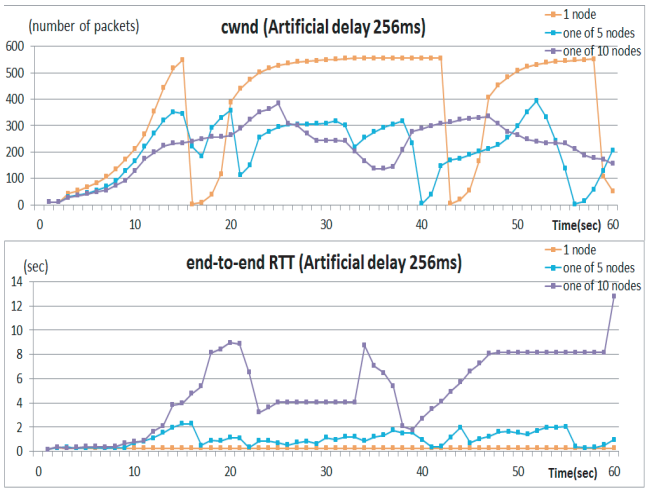

Figure 8. Transition of the CWND and RTT for an artificial delay of 256 ms

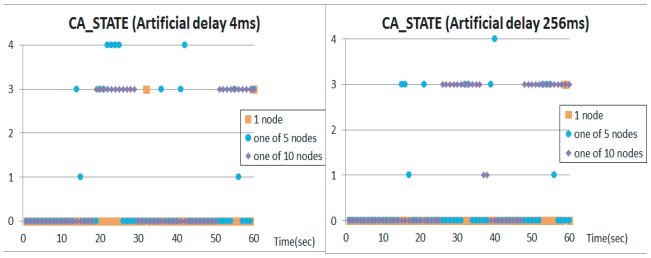

Figure 9. The state of TCP

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.

5.2 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 a RTT value in the experimental environment. The procedure is identical to that used in Section 5.1 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.

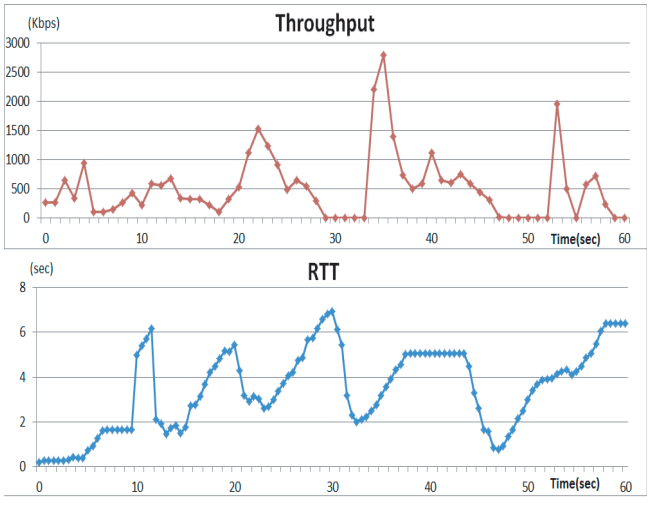

Figure 10. Time transition of the RTT and the Throughput

5.3 Case of proposed mechanism in all terminals

This subsection shows the effects of the proposed mechanism when all terminals utilize our middleware. The control parameters, the number of terminals and ratio-rtt, are set based on the value that was calculated referring to expressions shown in previous secition (Tables 2 and 3). In addition, we use a magnification value of ratio_rtt based on a delay level because the scales differ between the low- and high-delay environments. Thus, we divide a case into two phases according to min_rtt as follows.

A : $0 \leq \text{min\_rtt} < 100$

B : $\text{min\_rtt} \geq 100$

Table 2. Control parameters in the number of terminals

The number of terminals	A		B	
	max	min	max	min
1	90	80	555	250
2	80	60	400	300
3	60	50	300	200
4	40	30	200	120
5	25	20	65	40
6	20	10	55	30
7	12	8	39	20
8	8	5	28	15
9	7	3	24	10
10	4	2	17	5

Table 3. Control parameters in ratio_rtt

A			B		
ratio_rtt	max	min	ratio_rtt	max	min
1.0〜	100	80	1.0〜	555	300
10.0〜	80	60	2.0〜	300	100
15.0〜	60	50	3.0〜	100	20
20.0〜	50	40	4.0〜	10	5
25.0〜	40	30	5.0〜	8	4
30.0〜	30	20	6.0〜	6	3
35.0〜	20	10	7.0〜	5	3
40.0〜	10	5	8.0〜	4	2
45.0〜	4	3	9.0〜	3	2
50.0〜	3	2	10.0〜	2	1

Tables 2 and 3 are divided into two parts according to the two cases. In addition, we compare the control parameter decided by ratio_rtt (Table 3) with that decided by the number of the terminals (Table 2) and adopt the smaller one like the following algorithm.

```
if(max_cwnd(terminal) < max_cwnd(rtt)){
    max_cwnd = max_cwnd(terminal)
    }else{
    max_cwnd = max_cwnd(rtt)
    }
}

if(min_cwnd(terminal) < min_cwnd(rtt)){
    min_cwnd = min_cwnd(terminal)
    }else{
    min_cwnd = min_cwnd(rtt)
    }
}

min_cwnd < tune_cwnd < max_cwnd
```

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 the middleware. The performance was generally improved by introducing the middleware, regardless of the artificial delay time. Particularly, when many terminals communicate with an artificial delay of 256 ms, the performance improved by approximately 2 times.

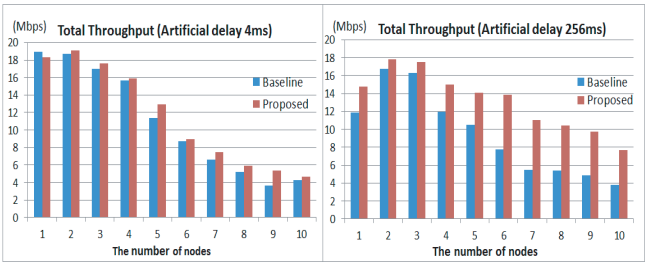

Figure 11. Total throughput

Examples of behavior the adjusted CWND when the artificial delay time is 256 ms is shown in Figure 12. An overall large delay can be evaded by each terminal that does not exceed the maximum CWND, which is set according to the number of terminals. Even if a delay occurs, the subsequent large delay is prevented by decreasing the CWND level depending on the ratio_rtt. This result can be confirmed by comparing the time transition of the default and adjusted RTT values, as shown in Figure 13. Moreover, almost no abnormal states including Fast Retransmission was detected as shown in Figure 14. Thus, the transmission rate is improved by using the appropriate control.

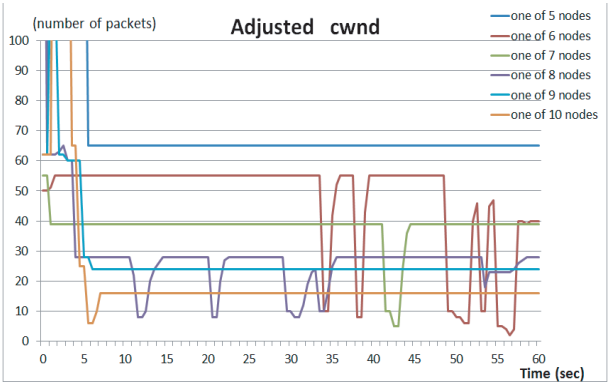

Figure 12. Examples of adjusted CWND

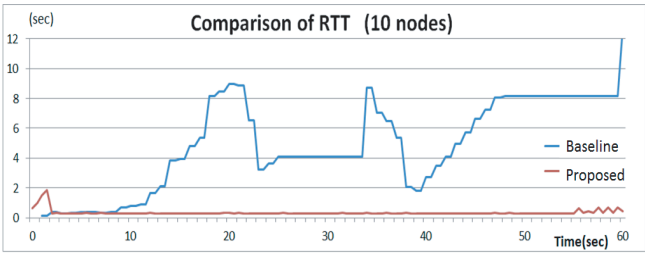

Figure 13. Time transition of RTT

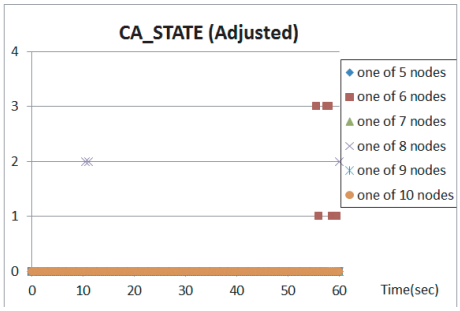

Figure 14. The state of TCP after adjusted

Next, we evaluate the communication performance using the Fairness Index [17] 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 5 shows the calculation method.

$$FairnessIndex: \ fi = \frac{(\sum_{i=i}^{k} x_i)^2}{k \sum_{i=i}^{k} {x_i}^2} \quad (1 \leq i \leq k) \qquad (5)$$

An evaluation of the fairness when the artificial delay is 256 ms is shown in Figure 15. The blue bar shows the baseline, and the red one shows the proposed mechanism. The Fairness Index keeps approximately 1 in many terminals of the proposed mechanism, whereas the fairness is spoiled when there is no middleware and the number of terminals increases. This result confirms that the equitableness is also improved by introducing the middleware.

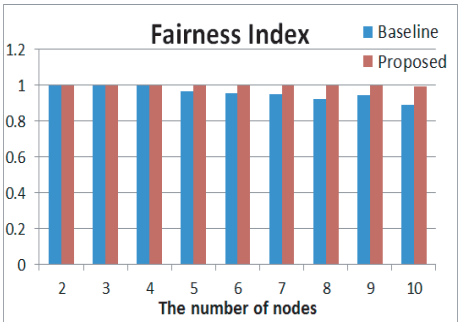

Figure 15. Fairness evaluation

According to the above results, we can successfully improve the transmission speed when many terminals communicate, particularly in a high-delay environment. The speed increases by approximately 2 times by introducing the suggested middleware. In addition, the fairness can be generally improved with the middleware.

5.4 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 Section 5.1, and the total number of terminals in WLAN is varied to 4, 6, and 8. Figures 16 through 18 show the throughput performance for the 4, 6, and 8 terminal cases, respectively. In these figures, the blue line shows the total throughput, the red bar indicates the average throughput of each terminal that utilizes our middleware, and the green one shows 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.

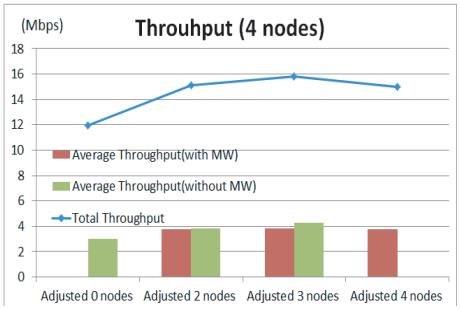

Figure 16. Throughput when 4 terminals communicate

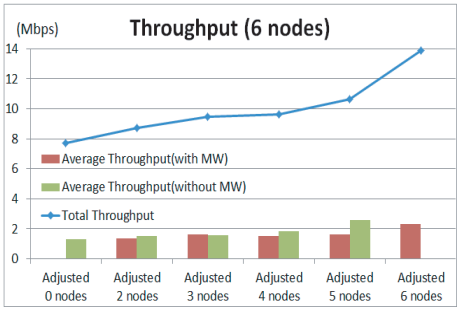

Figure 17. Throughput when 6 terminals communicate

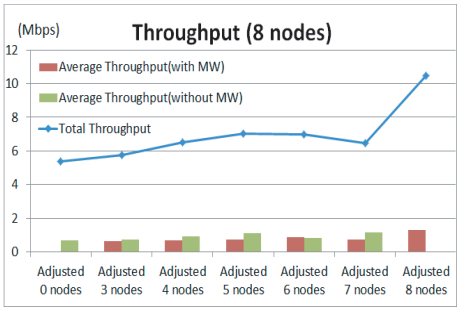

Figure 18. Throughput when 8 terminals communicate

5.5 Discussions

In the evaluations, the performance did not improve so much in the case of artificial delay of 4ms, as that of 256ms. This is because a BDP to control is originally too small. Anyway, it is necessary to build a system that automatically calculate a parameter in accord with the environment and control so that this suggestion technique is effective in all environments, because parameters shown in tables in Section 5.3 was tuned only for our environment.

In Section 5.4, we evaluated performance in the environment where terminals that did not have our middleware was 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 this study, we added the increase and decrease of RTT to a parameter and adjusted CWND. As a result of the comparison, the performance improvement of 2.6% - 41.8% was realized compared with the previous method, especially when the number of terminals is large.

Finally, all terminals access the same server in this environment, 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 changes 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, but such cases are discussed in other literatures and out of scope for our study.

6. 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.

An experimental study with up to 10 Android terminals shows that the proposed mechanism can improve both aggregate throughput and fairness of the WLAN. In particular, it improved the TCP throughput by a factor of 5 when many terminals are associated with the AP and very long RTTs are observed for the TCP sessions.

There are a few possible directions that can extend this study. The experiment presented in this paper has simply used a lookup table to determine appropriate CWND sizes, but this should be based on a mathematical model that can account for various conditions, including 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 off er 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.

Acknowledgement

The authors would like to thank to Dr. Mineo Takai at UCLA for a helpful advice to this work.

7. REFERENCES

[1] L. Xu, K. Harfoush, and I. Rhee, "Binary Increase Congestion Control for Fast, Long Distance Networks," Proceedings of Tech. Report, Computer Science Department, NC State University, 2003.

[2] Sangtae Ha, Injong Rhee and Lisong Xu "CUBIC: a new TCP-friendly high-speed TCP variant" ACM SIGOPS Operating Systems Review - Research and developments in the Linux kernel, vol.42, pp.64-74, July 2008.

[3] "A CSMA/CA MAC protocol of Cognitive Networks - emfield"

[4] Prasun Sinha, Thyagarajan Nandagopal, Narayanan Venkitaraman, Ragupathy Sivakumar, Vaduvur Bharghavan : "WTCP:a reliable transport protocol for wireless wide-area networks" Wireless Networks - Selected Papers from Mobicom'99 archive, Volume 8 Issue 2/3, March-May 2002, pp. 301-316

[5] Claudio Casetti, Mario Geria, Saverio Mascolo, M.Y.Sanadidi, Ren Wang: "TCP westwood: end-to-end congestion control for wired/wireless networks" Wireless Networks archive, Volume 8 Issue 5, September 2002, pp. 467 - 479

[6] Luigi A. Grieco and Saverio Mascolo: "Performance Evaluation and Comparison of Westwood+, New Reno, and Vegas TCP Congestion Control," ACM SIGCOMM Computer Communications Review, Vol.34, No.2, pp.25-38, April 2004.

[7] Shao Liu, Tamer Basar, R.Srikant: "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, Article No. 55.

[8] Hiromi Hirai, Saneyasu Yamaguchi, and Masato Oguchi: "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), pp.710-717, October 2013.

[9] Android open source project, http://source.android.com

[10] Android version share: http://developer.android.com/about/dashboards/index.html

[11] Saverio Mascolo, Claudio Casetti, Mario Gerla and M. Y. Sanadidi, Ren Wang: "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links", in Proc ACM SIGMOBILE 7/01 Rome, Italy , 2001.

[12] Kaori Miki, Saneyasu Yamaguchi, and Masato Oguchi: "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. 2011.

[13] W.Richard Stevens, TCP/IP illustrated, Vol.1 Protocol, Pearson Education, 2000

[14] The dummynet project, http://info.iet.unipi.it/ luigi/dummynet

[15] Iperf For Android Project in Distributed Systems, http://www.cs.technion.ac.il/ sakogan/DSL/2011/projects/iperf/

[16] Wireshark, http://www.wireshark.org/

[17] D.-M. Chiu and R. Jain, " Analysis of the increase and decrease algorithms for congestion avoidance in computer networks ", Computer Networks and ISDN Systems, vol. 17, pp. 1-14, 1989.

[18] Android Source Code: http://source.android.com/source/downloading.html

[19] Naoki Shiota, Hiroyuki Tomimori, Yoshiaki Okuyama, Shinichi Asai, Naoki Satoh: "Middleware for Controlling Multiple Bearer Utilization using Communication Policy" IPSJ SIG Technical Reports. MBL, [ISP SIG Mobile Computing and Ubiquitous Communication], 09196072, Information Technology Standards commission of Japan, no.44, pp7-12, May 2007.

[20] Yuki Tsuboi, Hitoshi Aida: "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, no.11, pp1-7, September 2011

[21] Dina Katabi, Mark Handley, and Charlie Rohrs: "Congestion Control for High Bandwidth-Delay Product Networks," In Proc. SIGCOMM'02, pp.89-102, August 2002.

[22] Vasilios A. Siris and George Stamatakis: "Optimal CWmin Selection for Achieving Proportional Fairness in Multi-Rate 802.11e WLANs: Test-bed Implementation and Evaluation," In Proc. WiNTECH'06, pp.41-48, September 2006.

[23] Dimitrios Koutsonikolas, Jagadeesh Dyaberi, Prashant Garimella, Sonia Fahmy, and Y. Charlie Hu: "On TCP Throughput and Window Size in a Multihop Wireless Network Testbed," In Proc. WiNTECH'07, pp.51-58, September 2007.

[24] Stevens, W.R.: "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," The Internet Society (RFC2001), 1997.