A Study on Transmission-Control Middleware on an Android Terminal in a WLAN Environment

Hiromi Hirai, Kaori Miki, Saneyasu Yamaguchi, and Masato Oguchi

Ochanomizu University, Department of Information Sciences,
Tokyo, Japan
{hiromi,kaori}@ogl.is.ocha.ac.jp
sane@cc.kogakuin.ac.jp
oguchi@computer.org
http://ogl.is.ocha.ac.jp/

Abstract. In this study, we present a transmission-control middleware, which enables an Android terminal to select a suitable TCP in a WLAN Environment. Various approaches toward developing a congestion control algorithm of TCP have been proposed to prevent congestion. Some of these approaches are loss based whereas others are delay based to predict network traffic, and their hybrid type also exists such as Compound-TCP[9] and TCPIllinois[10]. However, all of these approaches are designed to allow each terminal to run independently. Moreover, in the case of a mobile terminal, its TCP is limited to behave modestly to avoid filling the bandwidth. In this paper, we suggest a middleware that exchanges communication conditions to predict traffic on the basis of the number of communication terminals connected to the same access point. In the future, we will improve the middleware to predict the values of the Congestion Window of the other terminals.

Keywords: Middleware, TCP, Congestion control, Android

1 Introduction

In recent years, the digital convergence era has arrived. Computers are required to be convenient anywhere and anytime. The huge demand for portable computers produced the smartphone, a mobile computer. However, the architecture of the smartphone is different from that of the general-purpose PC because of the poor function of the I/O interface and the limited hardware of the mobile phone. To overcome this disadvantage, the smartphone is constantly connected to the Internet and obtains information through the Cloud. The proliferation of SNS services has also stimulated users to publish information with their smartphones. For instance, users can upload their videos to Youtube and synchronize their calendars or address books in the Cloud. Thus, the performance of the smartphone is measured not only by its processing power but also by its network availability. We focus on the networking system of the smartphone to improve upon it.

We adopt Android as the OS to be researched because it has the highest share in the international smartphone market. Moreover, Android's source code is open, and developers can freely customize or remodel the code.

The smartphone is a useful device that enables users to connect to the Internet, regardless of time and location. If a user uses a smartphone while traveling, the device connects the user from access point to access point. However, if a large number of smartphones connect to an access point at the same time, congestion will frequently occur.

In this study, we attempt to develop a middleware through which each internet device can share its own Congestion Window (CWND) to predict network traffic. To date, a collection of various TCPs have been developed. The previous TCPs predicted the traffic via packet loss or delay. All of them are designed for general-purpose PC to control the data flow independently. Android is one of OSs developed for mobile device. It is adopted TCP-cubic[1] and the highest value is limited so as not to interrupt own working system. TCP-cubic has loss-based congestion control, with which CWND is decreased drastically by a packet loss. In WLAN environment, packet loss does not always indicate congestion because of noise. Actually packet loss rate in WLAN is much higher than that in wired LAN. Nowadays, the technology of cloud is also growing as rapidly as that of smartphone. Smartphone often depends on cloud for processing. In this study, the case is assumed in which smartphone transmits huge packets to cloud server as shown Figure 1. In recent years, smartphone tends to enjoy mass rapid communications. Cloud service supplies reliable WAN to be accessed from any part of the world. Therefore, most of the packet loss takes place between smartphone and an access point. Moreover the value of CWND is important because RTT is expected high.

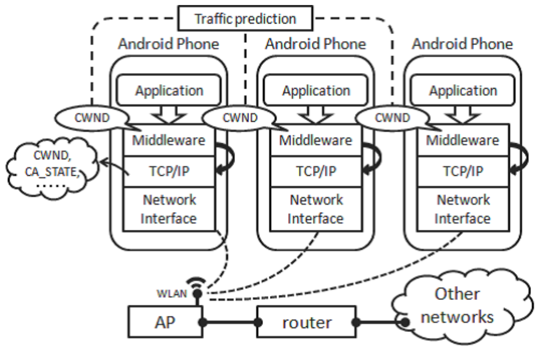

Fig. 1. The concept of our proposal

If only one Android terminal monopolizes an access point or base station without any obstacles, the terminal is supposed to maintain the highest communication throughput. Nevertheless, the terminals are scrambling against each other for bandwidth while a large number of nodes usually connect to an access point. In this case, the transport layer plays a critical role because it controls the data flow. CWND also indirectly influences the communication throughput in an Android terminal, as demonstrated by [14].

Additionally, in a crowded situation, CWND becomes insecure and is excessively restricted. CWND, which controls the amount of data segments to flow, is expected to increase as much as possible. However, a huge packet from a terminal whose CWND is excessively fixed may disturb the whole network and even interrupt the transmission with frequent packet losses and retransmissions [13]. Furthermore, no TCP algorithm always behaves appropriately [8], as any TCP has advantages and disadvantages.

Thus, TCP switching is important for each terminal because the terminals need to be able to adapt to their environments. Because the conventional transport layer for mobile terminals is designed to behave modestly to avoid jamming the network traffic, transmissions by mobile terminals have more latency. However, in recent years, mobile terminals have also come to enjoy rapid communication on a massive scale (i.e., an adequate mechanism is required to control mobile communication.) Therefore, we propose a transmission-control middleware to flexibly control TCP.

2 Android OS

2.1 Architecture

Android is a software stack for mobile devices that includes an operating system, middleware and key applications[2]. It was developed by the Open Handset Alliance (OHA), which is mainly composed of Google. As shown in Figure 2, Android, whose basement is Linux 2.6, has extra components for the smartphone and the tablet. Android typically has an original virtual machine or Dalvik VM.

As a Linux distribution, Android has the C library and is able to execute native code. However, the Dalvik binary written in Java is also useful on Android. Dalvic VM provides high-quality portability and security. Moreover, with the Dalvik library, which has various user interfaces and frameworks, users can enjoy the intuitive functions.

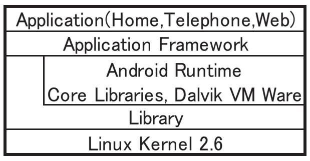
Application(Home,Telephone,Web)
Application Framework
Android Runtime
Core Libraries, Dalvik VM Ware
Library
Linux Kernel 2.6

Fig. 2. Architecture of Android

2.2 Application

The number of Android applications has increased rapidly for several reasons. First, both the package of the Android OS and the Android SDK used to develop applications are free. Furthermore, although they are free, their quality is still high. Second, the Android Market has enabled developers to spread Applications easily. Once an application is registered in the Android Market, it can be downloaded all over the world. The Android Market has also enabled users to obtain applications easily. Because of Dalvik VM, users only have to download a bytecodes to execute an application. Various applications can be easily installed to improve the usability of the smartphone itself. In this study, we understand the importance of applications on Android, although we focus on Android as a system platform. We improve the communication performance of Android through Dalvik VM.

3 Congestion control of Android

It is reported that initial CWND should be more than 10 to finish transmission quickly and that it interrupts no other communication[4]. We adopted this idea and set it 10.

3.1 Congestion control algorithm

As is widely known, the transport layer on the Linux OS plays a critical role because it controls the data flow. CWND is a parameter that limits the data flow to control network congestion. In the exact definition, CWND denotes the number of maximum packets that can be sent continuously without receiving acknowledgement from a data receiver. CWND also indirectly influences the communication throughput in the Android terminal, as demonstrated in a 2010 study[14]. In particular, in a high-delay environment, the impact is high.

CWND is directly related to the sender's transmission rate and should be set on the basis of the available bandwidth of an end-to-end connection so as to prevent network congestion. The default TCP of Android is TCP-cubic, which is an enhanced version of TCP-bic. Both of them have loss-based control to detect congestion. In the control, CWND is increased gradually per an Acknowledgement and halved each time a single packet loss is experienced. There are various error events of transmission, such as Local device congestion, Duplicate Acknowledgment, Selective Acknowledgment Options and Timeout. Especially in wireless LAN environment, packet loss is frequently caused by noise. Loss-based TCP assumes that packet loss happens only due to overflow of data segments. However, Packet loss does not always indicate congestion in wireless LAN environment. Actually delay-based TCPs, such as TCP-westwood and TCP-vegas, can fill more bandwidth than loss-based ones[5][6][7].

Previous study[13] improved TCP-cubic not to reduce CWND too much but to fill traffic with more segments. In this paper, we made experiment in switching TCP between TCP-cubic as default and original TCP. The original TCP is designed to adjust the sender's CWND high and transmit packets aggressively. It is argued that aggressive TCP might interrupt the others' transmission. However, it should be allowed for an smartphone to use it when no other devices are communicating.

3.2 How to obtain the parameters in Kernel

CWND is a parameter in Kernel. Kernel is a special software program that is different from other applications. Because it cannot accept normal debug methods, the behavior of kernel during communication is difficult to observe, even in the case of the general-purpose PC. This problem can be solved by using the Kernel Monitor[12].

In this paper, we applied the Kernel Monitor to Android, an embedded system. Kernel Monitor is our original system tool that can recode the value of each parameter in the kernel during the communication process. CWND, timestamps, the size of the socket buffer queue, error events and the other parameters are recordable. An overview of the Kernel Monitor is shown in Figure 3. The Kernel Moniter can leave a log of TCP behavior by inserting the monitor function into the source code of TCP and rebuilding the kernel.

Because Android is an embedded terminal, the amount of resources in Android terminals, such as storage and memory, are not sufficient to output the log that is inputed at the same time. To obtain a real-time log, we customized the Kernel Monitor to leave only the latest log and embedded it into the Android Kernel.

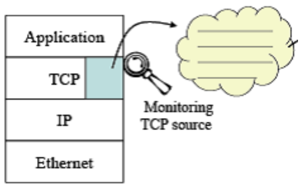

Fig. 3. Kernel monitor

3.3 Congestion in coexisting environment

If the network is crowded, packet loss often occurs. In this case, CWND is decreased drastically to prevent any more congestion. The transmission rate is also reduced at the same time. However, the present congestion control is running independently. If terminals increase their CWND and lose some packets at the same time, all of the terminals will decrease their CWND only to leave the traffic light.Furthermore, the TCPs of mobile terminals are designed to be modest in transmission. However, if only one terminal monopolizes an access point, it should be allowed to use more bandwidth. Therefore, synchronized congestion control must fill the bandwidth. In this study, we will offer not only TCP-friendliness but also fairness and effectiveness by providing synchronized congestion control in the future.

3.4 The design of transmission-control middleware

We propose a middleware that shares a communication condition among the other mobile terminals connected to an access point and selects a suitable TCP for the environment. Currently, common applications on smartphones are supported by Cloud services. Cloud services are protected in such a reliable network that the bottleneck of an end-to-end network lies around the WLAN access point. Therefore, traffic can be predicted more accurately by the other communication condition.

The middleware we present in this paper is composed of 3 parts. first one is an agent program written in C between application and Kernel Monitor. It is crosscompiled by GNU ARM tool chain[3]. It lets proc interface output the log left by Kernel Monitor and inform application of timestamp and CWND. Second one is a resident service activity that executes the agent program and get information about TCP to share among the other Android terminals during communication. Third one is a manager to predict the traffic and select suitable TCP. Android device is mobile and might be used in various environments. Thus we developed a middleware so as to keep more bandwidth and higher communication throughput.

4 The policy of switching TCP

4.1 Suitable congestion control algorithm

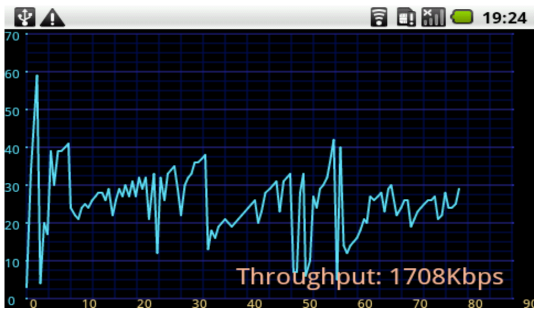

Fig. 4. Congestion control of default TCP (x-axis:time,y-axis:CWND)

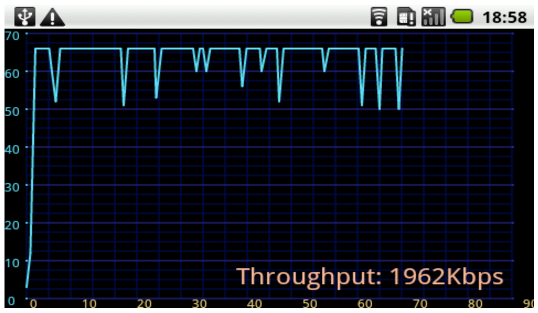

Fig. 5. Congestion control of original TCP (x-axis:time,y-axis:CWND)

TCP is designed for general purposes and does not always behave properly. Even if only one or two Android terminals are connected to an access point, the default TCP behaves modestly. In other words, it takes an accidental packet loss as congestion and decreases the CWND drastically if there is enough bandwidth. Default TCP means that TCP-cubic is initially embedded in the Android kernel.

In this study, the middleware observes the number of Android terminals connected to the same access point. If it notices that there is no other terminal, the middleware switches its TCP from default to original. If the middleware notices that some terminals are connecting to the access point, it switches its

TCP from original to default to avoid negatively affecting the other terminal's transmission.

In Figure 4 and 5, the experiment is conducted in Round Trip Time 128 ms and artificial packet loss 1%. These graphs are drawn on a TCP visualization tool as the author's graduation thesis. The figures show the behavior of the default TCP and that of the original one. In these figures, the horizontal axis and vertical axis denote time (seconds) and CWND, respectively. These congestion controls are observed if each terminal executes an application that transports a 16 Mbyte packet. Additionally, they are examined separately to monopolize the network.

Compared with the default TCP, the original TCP rapidly increases the CWND and decreases little even if several packets are lost. Moreover, low CWND is also recovered from quickly. It is reported that the original TCP is effective if RTT is longer than 64 ms. Therefore, the default TCP tends to excessively limit the CWND to retain high throughput.

5 Sharing of communication condition

Currently, most of the programs used to switch TCPs are working on Dalvik VM. We will embed them into the library as a middleware in the future.

5.1 An application on terminal

This application measures the latency taken to transmit some packets to measure the communication throughput and visualizes the congestion control.

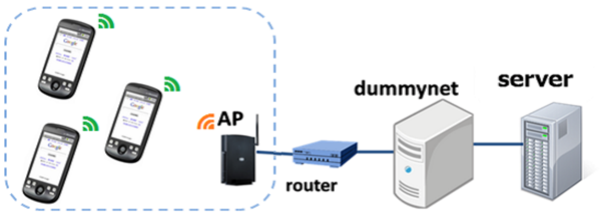

Fig. 6. the model of experiment

5.2 How the application obtains parameter in Kernel

In this experiment, we used two server programs on a PC. One is a receiver of packets from an experimental Android device. The other is an echo server that receives information about the communication condition and broadcasts it throughout the access point. If the receiver receives the whole packet, it replies with a sign. By comparing the time to start with the time of this sign, the application calculates the throughput. As shown in the figures of this application, the communication throughput of the transmission appears at the bottom.

However, while the connection is being established, the agent program is receiving its own TCP information left by the Kernel Monitor to analyze the TCP once per 800 ms in parallel. The function of accessing the Kernel Monitor is in an infinite loop with *sleep()*.

5.3 Exchanging CWND

The analysis of own TCP behavior is shown in section 5.2. We explain the sharing system in the next section.

Each terminal sends its own CWND by UDP after analysis. UDP is better to exchange each infomation because previous data are not required to retransmit in the case of packet loss. The information broadcasted from a terminal is caught by the other terminals' middlewares. Afterward, the middleware of the terminals considers whether it should switch TCPs. The visualization tool is also listening to the broadcasted information and drawing the graph on the basis of the data.

5.4 Experimental environment

The experimental environment is shown in table 1 and figure 6. FreeBSD was inserted between the access point and the server machine to work as Dummynet. The artificial delay is set at 128 seconds. The TCP behaviors of all of the terminals in an access point are visualized by using the original tool.

Table 1. The experimental environment

Android	Hardware	HT-03a
	Model number	AOSP on Sapphire(US)
	Firmware version	2.1-update1
	Baseband version	62.50S.20.17H_2.22.19.26I
	Kernel version	2.6.29-00481-ga8089eb-dirty
	Build number	aosp_sapphire_us-eng 2.1-update1
Server	CPU	Intel Pentium M 1.30GHz
	Main Memory	256MB
	OS	Linux2.6.32-31-generic
AP	Maker	BUFFALO
	Product name	WHR-G301N/U AirStation
	Mode	IEEE 802.11g

5.5 The detail about the experiment

In these experiments, three Android devices connect to the server through a WLAN access point. Whereas one terminal is transmitting 16 Mbyte of data, the other terminals start transmitting 4 Mbyte of data to interrupt the transmission. We conducted two experiments related to the transmission to compare non switching-TCP and switching-TCP.

In experiment 1, all of the Android terminals are transmitting with the default TCP. In contrast, in experiment 2, the middlewares are running to control the TCP in each section 4.1. That is, the main terminal transfers 16 Mbyte of data with the original TCP when it monopolizes the access point. Then, the main one switches its TCP from the original to the default when one of the other terminals begins transmitting. Finally, the main terminal changes its TCP from the default to the original when all of the others finish transmitting.

5.6 Experimental evaluation

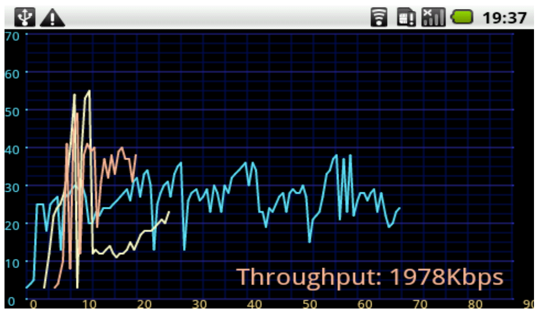

Fig. 7. The result of experiment 1 (x-axis:time,y-axis:CWND)

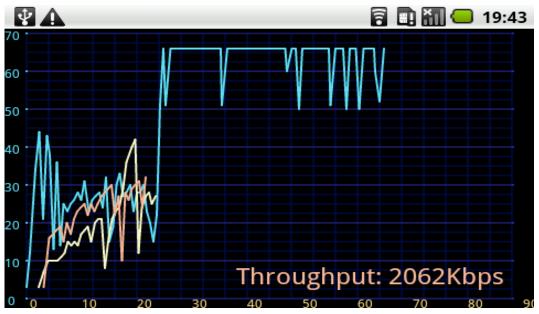

Fig. 8. The result of experiment 2 (x-axis:time,y-axis:CWND)

The results of these experiments are shown in Figure 7 and 8. In experiment 1, the main terminal behaved modestly, even though the other terminals did not communicate. The value of CWND was fluctuating during the whole time. In experiment 2, because of the switching TCP, the main terminal could transfer more packets to fill the rest of the bandwidth than it could in experiment 1 before the others had started transmitting and after they completed their transmissions.

The typical results are presented. No terminal showed the same behavior or communication throughput.

6 Conclusion and future works

In this paper, we showed the middleware that exchanges communication conditions in an access point. Communication throughput on the Android terminal is improved with the original TCP. Switching TCP is effective method because it allows the original TCP to run only if no other terminals are communicating. Overflow in traffic often has a negative impact on the other terminals or the network itself. Because the original TCP is aggressively designed, we must take care when using it.

We would like to improve this transmission-control middleware to fill the network traffic to the extent that congestion occurs. The number of communicating terminals is cared by the middleware in this paper. The middleware will be designed to consider the CWND of the other terminals in the future.

References

1. Sangtae Ha, Injong Rhee and Lisong Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant", SIGOPS Operating Systems Review, 42(5):64-74, 2008.
2. android developers: http://developer.android.com
3. Sourcery G++ Lite for ARM GNU/Linux: http://www.codesourcery.com/
4. Nandita Dukkipati, Tiziana Rece, Yuchung Cheng, Jerry Chu, Tom Herbert,Amit Agarwal, Arvind Jain, and Natalia Sutin, "An Argument for Increasing TCP 's Initial Congestion Window", in Proc. ACM SIGCOMM Computer Communications Review, vol. 40 ,No.3, pp. 27- 33,July2010.
5. 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.
6. Luigi A. Grieco and Saverio Mascolo, "Performance Evaluation and Comparison of Westwood+, New Reno, and Vegas TCP Congestion Control", in Proc. Computer Communication Review, 34(2), pp. 25-38, 2004.
7. Antonio Capone, Luigi Fratta and Fabio Martignon, "Bandwidth Estimation Schemes for TCP over Wireless Networks", IEEE TRANSACTIONS ON MOBILE COMPUTING, VOL. 3, NO. 2, APRIL-JUNE 2004.
8. Jiro Katto, Kazumine Ogura, Tomoki Fujikawa, Kazumi Kaneko and Zhou Su, "On Hybrid TCP Congestion Control", in Proc. ICCCS 2008, Nov.2008. ハイブリッド TCP
9. Alberto Blanc, Konstantin Avrachenkov, Denis Collange and Giovanni Neglia, "Compound TCP with Random Losses", in Proc NETWORKING '09 Proceedings of the 8th International IFIP-TC 6 Networking Conference Springer-Verlag Berlin, Heidelberg, 2009.
10. Shao Liu, Tamer Bas, ar and R. Srikant, "TCPIllinois: A Loss and DelayBased Congestion Control Algorithm for HighSpeed Networks", in Proc. VALUETOOLS (Pisa, Italy, October 2006).
11. Douglas J. Leith, Lachlan L. H. Andrew, Tom Quetchenbach, Robert N. Shorten, Kfir Lavi, "Experimental Evaluation of Delay/Loss-based TCP Congestion Control Algorithms", in Proc. the 6th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet 2008) , 5-7 March, 2008, Manchester, U.K.
12. Kaori Miki, Saneyasu Yamaguchi, and Masato Oguchi, "Kernel Monitor of Transport Layer Developed for AndroidWorking on Mobile Phone Terminals", in Proc. the Tenth International Conference on Networks (ICN2011), St. Seattle, U.S.A., April 2010.
13. Kaori Miki, Saneyasu Yamaguchi, and Masato Oguchi, "A Study about Performance Improvement and Analysis of Communication on Android in a Wireless LAN with Kernel Monitor", Multime-dia, Distributed, Cooperative, and Mobile Symposium (DICOMO2011), 7H-2, July 2011.
14. Hiromi Hirai, Kaori Miki, Saneyasu Yamaguchi, and Masato Oguchi, "A Visualization Tool for Communication Performance of Android Terminal", The 73rd National Convention of IPSJ, 5V-9, March 2011.