Kernel Monitor of Transport Layer Developed for Android Working on Mobile Phone Terminals

Kaori Miki
Masato Oguchi
Department of Information Science
Ochanomizu University
2-1-1, Otsuka, Bunkyo-ku, Tokyo, Japan
kaori@ogl.is.ocha.ac.jp
oguchi@computer.org

Saneyasu Yamaguchi
Kogakuin University
1-24-2 Nishi-shinjuku, Shinjuku-ku, Tokyo, Japan
sane@cc.kogakuin.ac.jp

***Abstract*—In recent years, with the rapid growth of smart phone market, Android is drawing an attention as software platform of embedded system, used as a personal digital assistance developed by Google. While Android is taken notice for its flexible development of application software and expansion of the system, we are interested in optimization and performance evaluation of network computing ability of Android. Because an embedded system like Android has architecture different from that of general-purpose PC, and due to the poor function of I/O interface, it is difficult to grasp what happens inside the embedded system precisely. Therefore, it is interesting to analyze the communication behavior of Android. In this paper, we have developed a Kernel Monitor tool suitable for an embedded system that is able to observe the behavior of kernel. We have applied this tool for the Transport layer of Android. We have shown that internal operation when an embedded system is communicating can be analyzed with our approach.**

***Keywords*-Android, Mobile Phone, Embedded system, Linux Kerne,**

I. INTRODUCTION

Recently, almost every user has own cellular phone. Moreover, it is not rare to have two or more cellular phones by a single user, and those phones are used in different ways depending on service and the usage. Previously, a cellular phone was used only for the voice call and text mail. However, since the transmission rate improves recently, it is applied to many functions including Internet access, distribution of music and animation, radio, television, IC card, and so on. Therefore, it is difficult to develop applications for a unique OS of each career due to its huge cost. Thus, commoditized OS for cellular phones has been desired. In this case, the basic part is shared as a platform, and original functions and services are developed individually. As a result, the efficient improvement of application development can be achieved. Moreover, since there is an advantage that it becomes easy for programmers to develop applications on it, the number of open software should be increased. Android [1] has been developed by Google for this purpose. Android is a software platform of an embedded system that works with portable devices.

Android is different from software used in current mobile phone terminals. There is no restriction for the application development because this is open source software. Applications can be executed on mobile phone equipped with Android regardless of the career or the device, and they can be highly customized. From these factors, the share of Android is increasing. While Android is drawing attention for flexible development of application software and expansion of the system, we are interested in Android as a system platform. In particular, since mobile phones, such as smart phones, become a leadingwe aim to optimize and evaluate performance of network computing ability of Android. Most of recent research works on Android concentrate only on applications, except [2] in which CPU load of Dalvik bytecode is investigated. Our research works are focused on the communication ability of Android platforms.

Because architecture of an embedded system is different from that of general-purpose PC, it is interesting to analyze its communication behavior. In particular, since the mobile phone such as smart phones becomes a leading part as a client terminal, to analyze their behavior is drawing attention in a variety of communication scenes. However, as for the behavior of the embedded system, the observation method is extremely limited due to the poor function of I/O interface. Because the resources of an embedded system is much less than that of general-purpose PC, the resource that can be spared for the monitoring and analysis is insufficient. Therefore, it is possibly considered having a substantial influence on the behavior of the system by its monitoring. From such a reason, it has been difficult to grasp exactly what happens in embedded systems during communications.

In this paper, since we have overcome the difficulty of an embedded system such as different interfaces and so on, we have developed a Kernel Monitor tool in Android, which has basically the same function of such a tool developed for general-purpose PC [3]. We have applied the tool in

 ISBN:978-1-61208-113-7

the Transport layer of Android. As a result, we show that the internal operation of an embedded system when it is communicating can be analyzed with this approach.

II. Development of Applications on Android

In this section, we explain about Architecture of Android and how to develop applications on it.

A. An Overview of Android

Table I
Architecture of Android

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

The architecture of Android is shown in Table 1. Android is constructed with Linux Kernel 2.6, and various components are added to its OS so as the platform to be composed. Because only the kernel part is adopted from Linux, it is possible to compose Android from various Linux packages.

Android Runtime, that is the application execution environment of original Android, is mounted on Linux kernel. The original virtual machine called Dalvik is installed in Android. This corresponds to Java Virtual Machine (JVM). Applications can be developed just suitable for Dalvik, because the application frameworks are provided on top of Dalvik for the execution of applications. Therefore the portability of Android is very good.

Android is different from other software implemented in current portable devices, and there is no restriction in the application development because this is open source software. Applications can be executed on mobile phone equipped with Android regardless of the career or the device, and they can be highly customized. Thus the load of the application development is considered to be reduced, and there is a flexible extendibility to another career and another model.

The communication is performed by using the protocol stack in Linux kernel. Thus, it is thought that the communication performance of Android is decided in this TCP implementation. Therefore, Transport layer in the kernel is highlighted and evaluated in this research work.

B. Cross Development

The cross development that uses a different computing environment is employed when applications of the embedded system is developed and executed. This is because mobile terminal's display is too small, and neither CPU performance nor memory capacity is enough. In Android, cross development is generally employed. The computer that develops chiefly is called a host environment, and Android terminal is called a target environment. The camera equipment and so on that the host environment does not usually include can be executed by the emulator inside hosts. Android is not suitable for development environment, because Android is an embedded system that has only limited commands compared with the case of general-purpose PC. Cross development can raise the efficiency of development.

Not only application development but also build of Android itself are executed on the form of the cross development. The Kernel Monitor introduced in this paper is also formed on the cross development, then implemented in the Android terminal.

III. Kernel Monitor

In this section, we explain Kernel Monitor which is our original system tool, and how to develop Kernel Monitor that works on Android terminal.

A. An Overview of Kernel Monitor

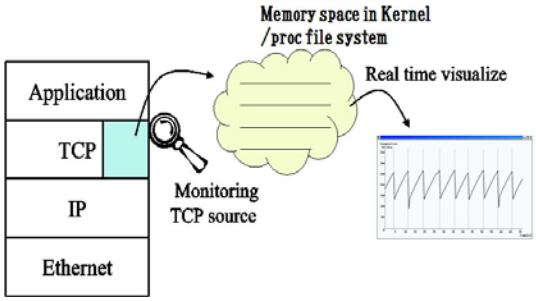

Figure 1. An Overview of Kernel Monitor

Kernel Monitor is a tool that can recode how the value of parameters in the kernel changed as a result by which part of the kernel code being executed at which time when communicating. An overview of Kernel Monitor is shown in Figure 1. We have inserted the monitor function in the Transmission Control Protocol(TCP) source code of the kernel, then recompiled the kernel, TCP parameters can be monitored as a result. Examples of what we can monitor with this tool are the value of Congestion Window (CWND) and various error events of communications (Local device congestion, duplicate Acknowledgment (ACK)/ Selective Acknowledgment Options (SACK), and Timeout). The behavior of kernel can be shown when it normally operates with the Kernel Monitor. In addition, when something wrong happens, it is possible to detect a specific problem and investigate what happens inside the kernel.

Kernel is special software different from other applications. It cannot accept normal debug methods, and therefore, it is difficult to observe the behavior of kernel during communications even in the case of general-purpose PC. However, in general-purpose PC, this problem has been solved by using Kernel Monitor [3]. In this paper, we

 ISBN:978-1-61208-113-7

have applied the Kernel Monitor to Android, an embedded system.

B. Development of Kernel Monitor for Android Terminals

Since the base of Android is Linux kernel, Android has a possibility to accept similar approach of general-purpose PC. However, as Android is an embedded system, it has a lot of different points from the case of general-purpose PC. For example, the amount of resources of Android terminals, such as storage and memory, are limited. Thus, the same approach as general-purpose PC may be impossible due to resource shortage. In addition, since the resource that can be spared for the operation analysis is insufficient, there is a possibility of having a substantial influence on the behavior of the system by the operation analysis itself. Android also need the cross compile for system and applications. OS is built in a special way. Moreover, it is required to use a special way to boot compiled OS on an Android mobile phone.

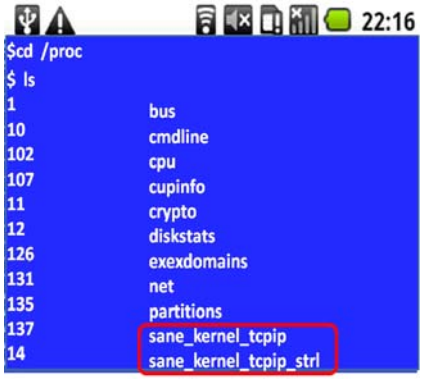

Figure 2. /proc file system

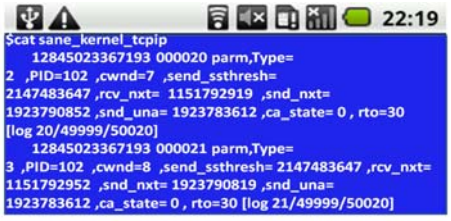

Figure 3. Log of Kernel Monitor

In this paper, we have overcome difficult problems peculiarly an embedded system's own, and developed Kernel Monitor that is the same as general-purpose PC basically. This Kernel Monitor has been implemented in Android OS codes, inserted into an Android mobile phone terminal, and we have confirmed that the Kernel Monitor runs on it. We show that it is possible to analyze the behavior of Android when it is communicating. Figure 2 and 3 are examples of captured display that Kernel Monitor is running on Android mobile phone. As a result, it becomes possible to show that Android's kernel behavior can be analyzed with almost the same way as the Kernel Monitor of general-purpose PC. Thus, we have realized the monitor tool in this environment.

As an example of monitoring, we have analyzed the relation between Congestion Window and throughput during the communication on Android. It is introduced in the following sections.

IV. Experimental System and Measuring Basic Performance of Android

In this section, measurement tool and experimental environment are shown, and the experimental way is introduced.

Table II
Experimental Environment

Android	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 ERE27
server	OS	Fedora release 10 (Cambridge)
	CPU	CPU : Intel(R) Pentium(R) 4 CPU 3.00GHz
	Main Memory	1GB

Table II, Figure 4 and Figure 6 show the experimental environment. In our study, we have cross-compiled iperf-2.0.4 [4], and inserted it Android mobile phone. With this tool, we have evaluated the socket access case as a basic performance. Arm-2008q3 [5] is used as a cross compiler.

A. Android to Android Communication Throughput

Figure 4. Android to Android Communication

First, we have evaluated Android mobile phone's throughput with IEEE 802.11g Wireless Local Area Network (LAN) through Access Point (AP) to another Android mobile phone, as shown in Figure 4.

Performance of socket access using TCP and User Datagram Protocol (UDP) is shown in Figure 5.

TCP communication average throughput of Android to Android is 8.2 (Mbps), while that of UDP communication is 6.7 (Mbps), as shown in the Figure 5. According to this graph, performance of UDP access is lower than that of TCP in the case of Android mobile phone terminals. There is almost no packet loss in both cases. The causes of throughput degradation seems to exist at the sender-side. The packet can be sent out only on a constant rate even if the

 ISBN:978-1-61208-113-7

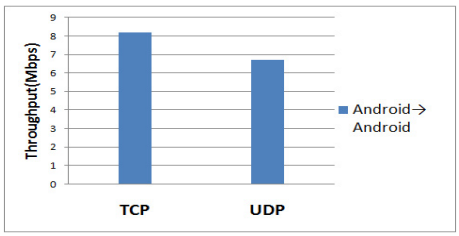

Figure 5. TCP thoughput of Android to Android Communication

bandwidth at the transmitting end is enlarged. Moreover, in UDP access, although we have not confirmed, performance of UDP access seems to be limited in the case of off-the-shelf devices. By way of comparison, we have evaluated Android-x86 that runs on x86 PC platform [6]. In this case, we have confirmed that throughput of UDP access is higher than that of TCP access.

B. Android Communication Performance of Remote Server Access

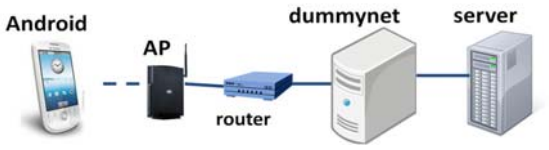

Figure 6. Android Communication Performance of Remote Server Access

We have evaluated Android communication performance of remote server access with dummynet (see Figure 6), which artificially generates delay. This supposes to access to a server on a remote place, which offers mobile cloud service.

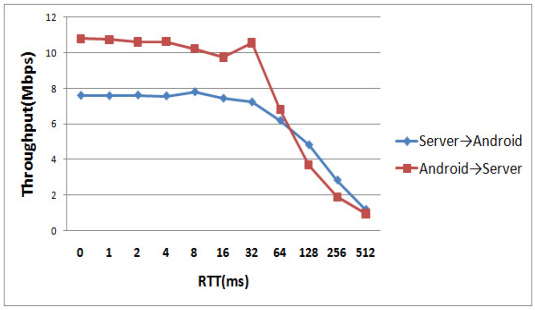

Figure 7. TCP Throughput between Server and Android Terminal in a Higher-Latency Environment

Figure 7 is a graph of throughput in which the horizontal (x-)axis is Round Trip Time (RTT) by dummynet. Better throughput is observed in the case that server is used as a receiver-side, because server's receive buffer is larger than that of Android terminal. However, in a higher latency environment, the performance in which server is receiver is declined. This is because Android cannot increase CWND enough in a higher-latency environment, and therefore CWND is running out. We have applied Kernel Monitor developed for Android, and analyzed CWND. It is saturated at 66: with this value CWND is not enough. Further details about Kernel Monitor is explained in the next section.

V. TCP TUNING AND APPLY KERNEL MONITOR TO ANDROID

We have tried to evaluate the effectiveness of changing Congestion Window control algorithm in TCP/ Internet Protocol (IP). We call it TCP Tuning in the rest of this paper. Two Congestion Window control algorithms are available for Android — Reno and Cubic (default). One of the most suitable Congestion Window control algorithm for mobile terminal is Westwood that tolerates packet loss, which cannot be applied to Android in this case. This depends on downloaded source code of Android when it is built.

A. Congestion Control Algorithms

A lot of congestion control algorithms are discussed in the literatures [7][8]. Reno is the basic algorithm. A wide variety of algorithm has been developed based on Reno. Reno detects congestion by packet loss and it regards transfer rate at the time as available bandwidth. For example, if three consecutive Duplicate ACKs are received, Reno regards it as occurring of packet loss and reduces CWND by half. Moreover, it increases CWND on receiving every ACK. Thus, Reno is an algorithm that increases the size of CWND gradually and drops it by detecting congestion. CUBIC [9][10] is an improved algorithms of Binary Increase Congestion Control (BIC). BIC is an algorithm that in normal TCP congestion control, linear search is executed for available bandwidth, on the other hand, binary search is executed in BIC. Window size of CUBIC is increased gradually.

B. Result of Android TCP Tuning

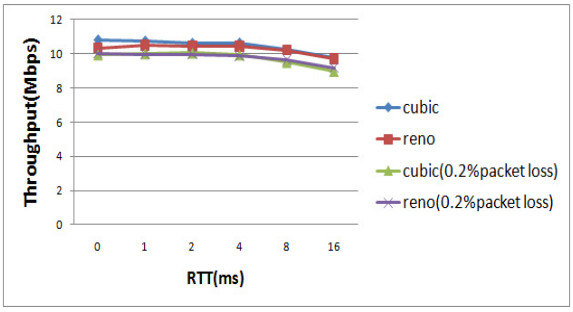

Figure 8. TCP Throughput between Server and Android Terminal with Various Algorithms

Figure 8 shows the result of performance that each Congestion Window control algorithms are applied to Android.

 ISBN:978-1-61208-113-7

Packets are sent from Android terminal to server. The performance of a round-trip including 0.2% packet loss nearly equal to that of a round-trip with no packet loss. The performance of the case including packet loss has a little decline, although there are no substantial difference between them.

C. Performance of Android to Android Communication

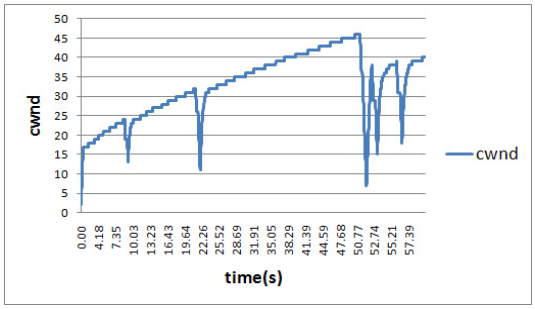

Figure 9. Cubic in RTT=0 [ms] including 0.2% packet loss.

CWND of Android to Android communication is shown in Figure 9 with basic communication by Kernel Monitor. Figure 9 shows that CWND degrades sometimes during the communication. Since the receiver side is Android terminal in this experiment, this is considered to be resource shortage in some cases.

D. Comparison of CWND size in TCP Tuning

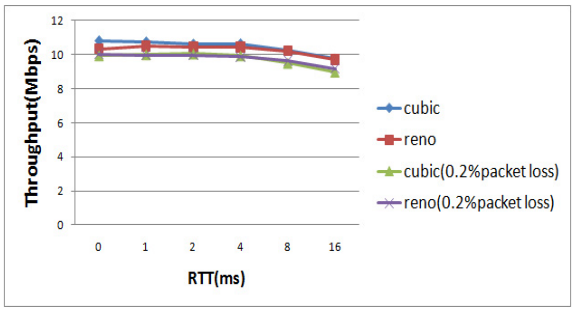

Figure 10. TCP Throughput between Server and Android Terminal with Various Algorithms

Next, we have observed the difference of CWND behavior in the case of TCP Tuning with Kernel Monitor. Figure 10 shows the behavior of CWND with Cubic in RTT=0[ms] including 0.2% packet loss. Figure 11 shows CWND with Reno.

According to Figure 10 and 11, we can observe the way of increasing CWND size is different depending on the Congestion Window control algorithms. Comparing both figures, Reno sets CWND size at half of ssthresh when congestion occurs. Next time when congestion occurs, Reno

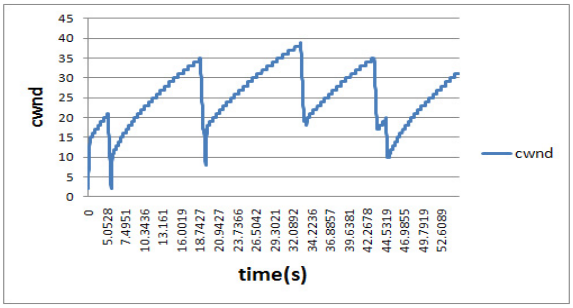

Figure 11. Reno in RTT=0 [ms] including 0.2% packet loss.

restarts CWND size at ssthresh and enters to congestion avoidance phase. Thus, Reno reduces CWND size less than that of Cubic when CWND size is decreased. The way of increasing CWND size is different from another one, although maximum value of CWND is between 40 and 45 in both cases. Throughputs of these algorithms are almost equal. In the case including packet loss, the behavior of CWND can be observed differently from another algorithm.

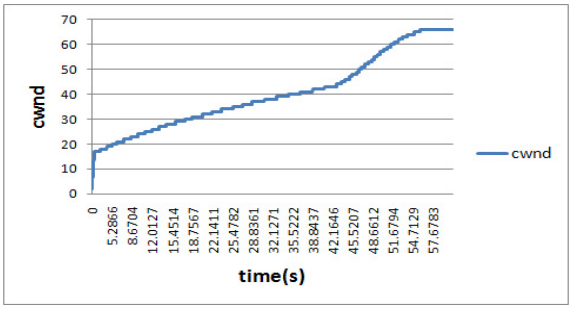

Figure 12. CWND value with Cubic in RTT=0 [ms]

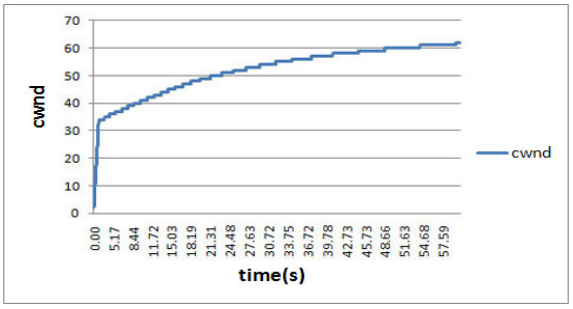

Figure 13. CWND value with Reno in RTT=0 [ms]

Next, the behavior of CWND in both cases of TCP Tuning is compared without packet loss. Although there were no substantial difference between two algorithms without packet loss in our previous work monitoring on Android-x86 [6], Figure 12 and 13 show that the way of increasing CWND is different from each other. This is because Android

mobile phone terminal increase CWND slower than the case of Android-x86.

VI. Conclusions

In this paper, we have developed Kernel Monitor that works on Android terminals. As a result, it becomes possible that kernel of Android can be analyzed with almost the same way as that of the Kernel Monitor of general-purpose PC. In addition, with Kernel Monitor, we have analyzed the relation of throughput and CWND behavior of Android during the communication.

As a future work, we are going to analyze various parameters of Android with Kernel Monitor, find peculiarity of Android, and analyze that behavior in detail. Especially, we are going to investigate into the behavior of simultaneous communication by multiple Android terminals.

Moreover, since most of Android communication is performed by applications on Dalvik, we are going to analyze the specific feature of Dalvik.

References

[1] Android:http://www.google.co.jp/mobile/android

[2] Takashi Majima, Tetsuo Yokoyama, Gang Zeng, Takeshi Kamiyama, Hiroyuki Tomiyama, and Hiroaki Takeda, ”CPU Load Analysis Using Dalvik Bytecode on Android,” IPSJ SIG Technical Reports, March 2010

[3] Reika Higa, Kosuke Matsubara, Takao Okamawari, Saneyasu Yamaguchi, and Masato Oguchi, ”Analytical System Tools for iSCSI Remote Storage Access and Performance Improvement by Optimization with the Tools, ”In the 3rd IEEE International Symposium on Advanced Networks and Telecommunication Systems (ANTS2009), December 2009.

[4] Iperf:http://downloads.sourceforge.net/project/iperf/iperf/2.0.4

[5] Sourcery G++ Lite 2008q-3-72 for ARM GNU/Linux:http://www. codesourcery.com/, http://www.codesourcery.com/sgpp/lite/arm /portal/release644

[6] Miki Kaori, Masato Oguchi, Saneyasu Yamaguchi, ”A study about behavior of Transport layer on Android terminals in a wireless LAN,” In Summer United Workshops on Parallel, Distributed and Cooperative Processing (SWoPP2010),August 2010.

[7] Kojima Akihisa, Ishihara Susumu, ”A Congestion Control of Cooperate with Internetmediate Node for MANET” Multimedia, Distributed, Cooperative, and Mobile Symposium (DICOMO2007),July 2007.

[8] Hasegawa Go, Murata Masayuki, ”Transport-layer protocols for high-speed and log-delay networks” The Institute of Electronics, Enformation and Communication Engineers, Technical Committee Conferences,February 2007.

[9] Sangtae Ha, Injong Rhee, and Lisong Xu, ”CUBIC: A New TCP-Friendly High-Speed TCP Variant” ACM SIGOPS Operating Systems Review, Volume 42 Issue 5, pp.64-74, July 2008.

[10] Habibullah Jamal and Kiran Sultan, ”Performance Analysis of TCP Congestion Control Algorithms” International Journal of Computers and Communications, Issue 1, Volume 2, pp.30-38, 2008.

 ISBN:978-1-61208-113-7