An Empirical Study on Applying Anomaly Detection Technique to Detecting Software and Communication Failures in Mobile Data Communication Services

Hiroyuki Shinbo, Toru Hasegawa

An Empirical Study on Applying Anomaly Detection Technique to Detecting Software and Communication Failures in Mobile Data Communication Services

Hiroyuki Shinbo and Toru Hasegawa

KDDI R&D Laboratories Inc.
2-1-15 Ohara, Fujimino-shi, Saitama 356-8502, Japan
{shinbo,hasegawa}@kddilabs.jp

Abstract. A mobile operator offers many mobile data communication services to its users, such as e-mail, Web browsing, company proprietary services. Although quick detection of communication and software failures are important to improve users' satisfaction, such a quick detection is difficult because the services are served by many servers, network nodes and mobile terminals. Thus we developed the anomaly detection technique for the mobile operator's network to detect anomalies caused by communication failures such as server and network halts. Our technique is based on the observation that users reconnect to servers many times when a communication failure occurs. It is useful not only to detect such communication failures, but also those which would be caused by software failures of mobile terminals and servers. This means that a mobile operator would be able to detect software failures missed at the testing period. In this paper, we empirically study how our technique is used to detect software failures of mobile terminals.

Keywords: mobile data communication, anomaly detection, interoperability testing

1 Introduction

Mobile terminals such as cellular and smart phones are owned by most people, and mobile data communication services such as e-mail and Web browsing services are becoming inevitable tools for social life. Since out-of-service has a serious impact on it, failures leading to out-of-service should occur as less frequently as possible. However, complete prevention is difficult due to a complicated system structure providing such a service. First, it is prone that software failures, i.e. bugs, are overlooked even by intensive tests because the system consists of various programs which run on various servers and mobile terminals. Especially, due to the competition in the mobile communication market, the number of mobile terminal models is becoming larger and a development period including a testing period is becoming shorter. Second, a service request is not always completed because of insufficient resources of servers and a network. It means that

some service requests are thrown away by a congested server and that some messages are lost at a congested link. Of course, such an incomplete service request is also caused by hardware failures of servers and network equipments. Please note that the failures caused by problems on wireless links are out of scope in this paper. Our motivation is to detect failures that affect around a mobile data communication service, and it does not include detections of failures caused by each wireless environment of mobile terminal.

Quick detection of such failures in an operational network is a practical and promising approach. This approach is called anomaly detection [1] assuming that such a failure exhibits some unusual behavior (This unusual behavior is called anomaly.). We have developed an anomaly detection tool to detect how users abruptly change their behaviors [2] regarding a reconnect as the fact that a user's service request is not successfully completed due to some failure. The tool monitors service request messages on the network and calculates how many users reconnect to servers in every sample period, e.g., 180 seconds. Then, it detects an anomaly when a reconnecting terminal ratio (the ratio of terminals reconnecting to a server to all terminals) of current sample period abruptly changes from the previous one.

We applied this tool to a commercial mobile data communication system [2] and the results show that it can detect failures which result in an abrupt increase of reconnecting terminal ratio. Example failures are halts of components which simultaneously handle many sessions such as servers and network equipments. (A session is a communication path between a mobile terminal and an application server. It corresponds to a single service request.) These failures make many users simultaneously reconnect to servers. However, the tool may miss failures if only a small portion of users reconnects. We think that software failures would fall into this category of failures. (In this paper, we call a software failure as a "*bug*", and it does not mean problem locations within program codes in softwares.) For example, some bugs may happen in limited conditions. Other bugs exist in only software programs running on some specific model.

The goal of this paper is to empirically understand how such bugs are detected using our anomaly detection technique. Our insight is that if we focus on only mobile terminals or servers which have a bug, the reconnecting terminal ratio of them abruptly changes. For example, if a new release of software programs is affected by a bug, the reconnecting terminal ratio of mobile terminals which downloaded it would increase. Thus we calculate the ratio with some group of mobile terminals in order to know whether a bug which was overlooked in the testing period is detected or not.

The contributions of the paper are three-fold. First, we actually found a bug of mobile terminals as an anomaly by analyzing the log data of a mobile data communication system. This implies that appropriate grouping of mobile terminals enables to detect a bug of mobile terminals which have a common feature, e.g., the same release and the same application. Second, a threshold used for detecting anomaly is carefully determined. Third, our anomaly detection tool

is so scalable that a few PCs (Personal Computer) with the tool can monitor a commercial mobile data communication system.

Although this paper does not deal with software testing techniques, the anomaly detection technique which this paper proposes is complementary to these techniques and plays an important role to detect bugs as soon as possible. This paper is organized as follows. Section 2 provides an overview of a mobile data communication system. Section 3 describes our anomaly detection tool and its algorithm. Section 4 describes how this tool is applied to detect anomalies caused by mobile terminals' bugs. Section 5 discusses the related work. Section 6 presents our conclusions.

2 Overview of mobile data communication system

A mobile operator offers many mobile data communication services such as e-mail, Web browsing and company proprietary applications as shown in Fig.1. We call a mobile data communication service just as a "*service*" and a mobile data communication system just as a "*system*" in the rest of this paper.

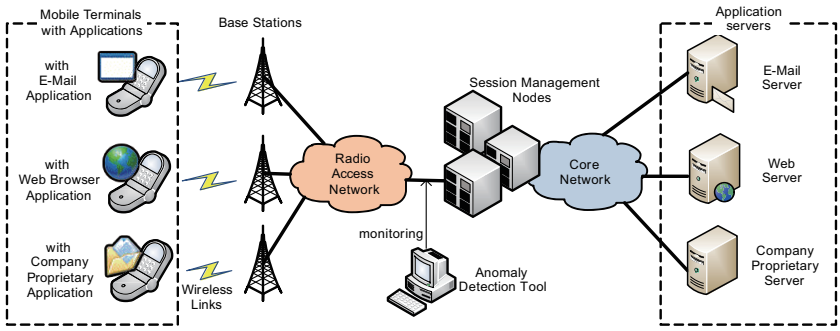

Fig. 1. Overview of a mobile data communication system

A system consists of the following components:

- *Applications* are software programs which provide a service to a user and they are running on a mobile terminal and an application server. A service is provided to a user by combination of applications on both the mobile terminal and server.
- *Mobile Terminals* are accommodated by a mobile operator's *Base Stations* and *Radio Access Network*.
- *Session Management Nodes* provide authorization and charging functions for services. After a mobile terminal is authorized by a session management node by sending a session creation request message, it can send a data request to an application server.

– *Application servers* are operated by either the mobile operator or third party application providers. The servers are accommodated by a *Core Network*.
– *Anomaly Detection Tool* detects anomalies based on a traffic monitoring, and it was developed by us. The detail will be described and discussed in Sect. 3 and later.

Figure 2 shows how a user gets a service in the following two steps.

(A) Session creation step
A *session* is a communication path between a mobile terminal and an application server. Before a user gets a service, a mobile terminal should send a *session creation request* message to a session management node. The mobile operator authorizes the user (or the mobile terminal) by validating this message. After the authorization succeeds, a *session creation complete* message is sent back and a session is created at the session management node. The mobile operator can charge user-data which is sent by the authorized mobile terminal based on the created session. The session creation request message includes the information of identifications of the mobile terminal and the requested service.

(B) Data communication step
After the session is created, *data communication* starts. The application on the mobile terminal sends *data request* messages to an application server and it sends back a data response message. This communication is a request-response style. In the case of a Web browsing service, they correspond to a HTTP Get request message and a HTTP Get reply message containing the data.

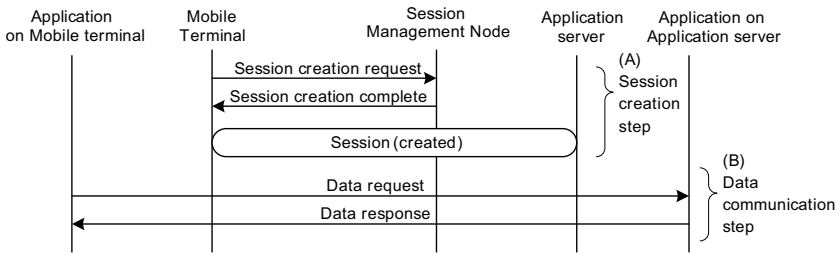

Fig. 2. Steps to get a service

3 Methodology

3.1 Principles of anomaly detection

An anomaly is an unusual behavior of some component of the system which is likely to be caused by a failure. Our motivation is to use the detected anomaly

to identify a failure causing it, thus we summarize failures before defining the anomaly. Table 1 lists possible failures for individual components in the system, such as mobile terminals and session management nodes. Please note that all failures are not listed and that these failures are typical ones. Possible failures are categorized to the steps in Fig.2:

- (A) Session creation step: A failure of this step results in session not being created.
- (B) Data communication step: A failure of this step results in data request not being completed, even if a session is successfully created.

Table 1. Possible failures of the steps (A) and (B)

Step	Component	Possible failures
(A)	Mobile terminals	Hardware failure, Protocol software program bug
(B)		Application software program bug
(A)	Session management nodes	Hardware failure, Node software program bug, Overload
(B)	Application servers	Hardware failure, Application software program bug, Overload
(A)	Radio access network	Network congestion, Network equipment (e.g. Ethernet switches, Routers) failure, Link failure
(B)	Core network	

Usually, an abrupt change of the number of messages (packets) is defined as an anomaly in the Internet [1]. Thus in order to identify which component is in failure, all messages sent by all components such as mobile terminals, session management nodes and application servers should be monitored. It means that many tools (equipments with network interface cards) capturing messages should be set at many links to which these components are connected. Although the method of monitoring all messages is useful to precisely identify the failed component, applying many tools is too expensive to be used commercially.

On the contrary, we use as small number of tools as possible. As described in Sect. 3.2, a few anomaly detection tools are set at the link connected to session management nodes as shown in Fig.1. The anomaly detection tool captures session creation request messages. Apparently, failures at the session creation step (A) are easily detected because the tool can monitor failed session creation request messages. On the contrary, how anomalies caused by failed data requests are detected is an important issue.

We focus on the observation that a user (a mobile terminal) reconnects to an application server after a mobile data communication service is not successfully completed [3]. It means that since a user should create a session before getting a service from an application server, the user re-sends a session creation

request message to a session management node again. Such a re-sent session creation request message is regarded as the fact that a user reconnects to an application server. Thus we define *reconnections* as mobile terminals which re-send session creation request messages for reconnecting to an application server, and an *anomaly* as an abrupt increase (change) of reconnections. (The anomaly is precisely defined in Sect. 3.3.) This enables to detect a failure of a data request to an application server at the data communication step (B) without capturing data request and response messages. As far as we know, only our tool uses such reconnections to detect anomalies in other communication systems.

This definition has two advantages. First, capturing only session creation request messages requires less computing power. It is more scalable than capturing all messages sent by all components such as mobile terminals, session management nodes and application servers. Second, this apparently reflects customers' satisfaction.

3.2 Anomaly detection tool

- The anomaly detection tool takes a traffic monitoring approach rather than a probing approach [1].
  - In a probing approach, a probing tool sends test messages to individual components in the system and thus it clearly pinpoints a failure of each component. However, it is time-consuming because a mobile data communication system consists a number of components, e.g., more than hundreds of servers.
  - A traffic monitoring approach enables to quickly detect anomalies because captured packets (messages) are immediately analyzed just after capturing them. In addition, this traffic monitoring avoids imposing load on servers and network nodes in the system.
- The anomaly detection tool is a software program running on a PC (Personal Computer) with a network interface card. The PC with the implemented tool is set at a link of a session management node in a radio access network (described in Fig.1), and it is used to capture only session creation request messages transferred between mobile terminals and session management nodes. The implemented tool running on a PC with Intel CoreDuo 2.0GHz CPU and 2G bytes memory can simultaneously process more than 60,000 session creation request messages per a minute. Thus a few anomaly detection tools are enough to monitor all sessions in a commercial mobile data communication system [2].

3.3 Anomaly detection algorithm

This section precisely defines the metric for the anomaly detection. We defined the anomaly as an abrupt change of reconnections, the metric is based on how many mobile terminals re-send session creation request message. Before defining it, we define how many times a mobile terminal sends session creation request messages in a sample period a "*session count*". Since a session creation request

message does not explicitly specifies that it is a reconnection, we regard those subsequent to the first request as reconnections. That is, if the session count is 2, the number of reconnections is 1.

An important issue is whether sending of these session creation request messages are really reconnections. Thus we carefully determine how long the sample period is so that no new session creation request messages exist in the sample period. In addition, the sample period should be determined as a small value as possible for quickly anomaly detections.

We collected the 2 month's log of session creation request messages in the system and investigated session intervals of each mobile terminal without failures. A session interval is defined as an interval of sending successive session creation request messages. Then, a cumulative frequency distribution of session intervals is created. We see that 90% of intervals are more than 180 seconds. Then we determine 180 seconds as the sample period. Thus in a 180 seconds sample period, about 90% of session creation request messages might not be reconnections. In this case, about 10% of session creation request messages might be reconnections. We investigated the number of reconnections in a 180 seconds sample period and obtained that about 90% of the number of reconnections was once. (i.e., the session count is 2.)

Another important issue is for what group of mobile terminals and servers the metric is calculated. A mobile operator or a third-party application provider is responsible for each service. Thus the metric is calculated from session creation request messages to the same server which corresponds to a service. Before defining the metric, we calculate a distribution of how many terminals connects to a server the "*session count*" times. We define $v_m[n]$ as the number of mobile terminals which have the session count n at the sample period m. Figure 3 shows how to get a session count for mobile terminals. In Fig.3, at the sample period m, the five mobile terminals *MT-A* to *MT-E* send session creation request messages. 2 (*MT-B* and *MT-E*), 2 (*MT-C* and *MT-D*) and 1 (*MT-A*) terminals send 1, 2, 3 session creation request message(s), respectively. That is, the results of $v_m[n]$ are $v_m[1] = 2$, $v_m[2] = 2$ and $v_m[3] = 1$.

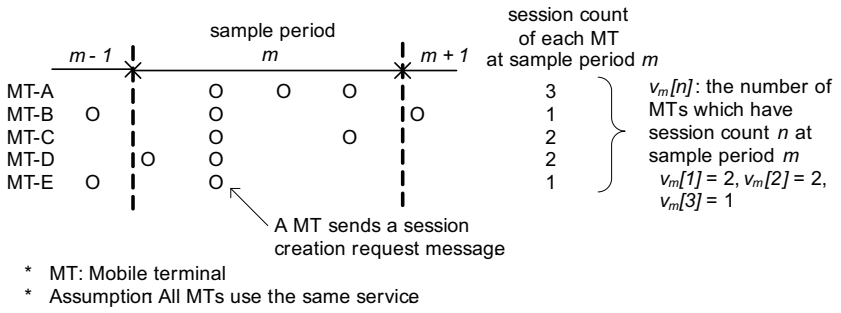

Fig. 3. How to get a session count

The anomaly detection algorithm focuses on mobile terminals which reconnect once and it means that values of $v_m[2]$ are used. Please note that since about 90% of the number of reconnections was once in a 180 seconds sample period as described at the above, in the rest of paper, we use 2 as a session count n and omit it from variable names. The other session counts or the sum of a several session count values can be chosen.

After this, we will explain how to detect an anomaly based on $v_m[2]$. Precisely, it uses how x_m is changed from that of the previous sample period $m-1$. x_m is the normalized value of $v_m[2]$ with respect to sum of $v_m[n]$ as shown by Equation(1). In Fig.3, x_m is calculated as $v_m[2]/(v_m[1] + v_m[2] + v_m[3]) = 0.4$. We call x_m as the "*reconnecting terminal ratio*" and it is the metric for anomaly detection. A reason why the normalized value used for the metric is that it is not sensitive to a change of the number of total sessions.

Before explaining how to use x_m for an anomaly detection, we defines two values y_m and y'_m. y_m in Equation(2) is the exponential average [4] of x_m, and y'_m in Equation(3) is the square of exponential average of x_m. The α is used for calculating exponential average in Equation(2) and (3). To decide the α, we need to decide the time-window tw and the sample period p. The time-window means that x_m values before tw get lost in oblivion, and we decide one day as tw (one day equals to 86,400 seconds). p is defined as 180 seconds from the beginning of this section, and thus α is calculated as around 0.002 ($= p/tw$).

Equation(4), (5), (6) and (7) are used for the anomaly detection based on x_m. We define the condition of an anomaly detection as that a difference g_m is more than a threshold t_m shown as Equation(7).

- g_m is a difference between the current x_m and the exponential average of y_{m-1}.
- We choose the threshold t_m based on the standard deviation σ_m. Equation(5) shows that σ_m is calculated using the exponential average y_m and y'_m calculated by Equation(2) and (3). The threshold t_m is calculated by k times as the standard deviation σ_m shown as Equation(6). The standard deviation σ_m means a possible range of a difference between x_m and the exponential average y_{m-1} in normal case at the sample period m. Our algorithm detects an anomaly when the difference g_m is more than k times of the possible range.

$$x_m = \frac{v_m[2]}{\sum_{i=1,\infty} v_m[i]} \tag{1}$$

$$y_m = \alpha \times x_m + (1-\alpha) \times y_{m-1} \tag{2}$$

$$y'_m = \alpha \times x_m^2 + (1-\alpha) \times y'_{m-1} \tag{3}$$

$$\text{where} \quad \alpha = p/tw$$

$$g_m = abs(x_m - y_{m-1}) \tag{4}$$

$$\sigma_m = \sqrt{(y'_m - y_m^2)} \tag{5}$$

$$t_m = k \times \sigma_{m-1} \tag{6}$$

$$g_m > t_m \tag{7}$$

Figure 4 shows how anomalies are detected by Equation(7). It shows a time series of 600 samples of g_m and each circle is g_m in Equation(4) at each sample period. It also shows the curve which plots values of t_m in Equation(6) where k is 3. Samples which are over the curve are regarded as anomalies, and they are marked by cross marks in Fig.4. We will discuss how k is determined in Sect. 4 based on the log data.

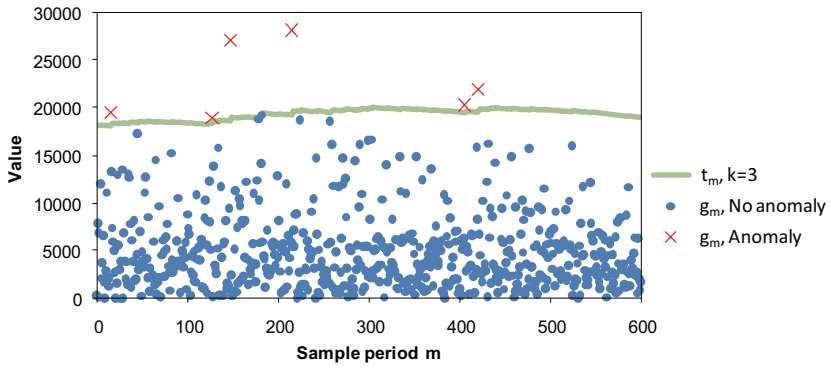

Fig. 4. Anomaly detection example

3.4 Applying anomaly detection tool to detecting bugs

It is relatively easy to detect anomalies caused by failures of session management nodes, servers, a radio access network and a core network because these failures result in many session creation failures. However, we consider that detecting anomalies caused by a bug (software failure) of mobile terminals is not easy because all mobile terminals are not always affected by this bug. For example, when some bug is injected to an application software program which is installed in a new release of mobile terminals, only such mobile terminals have the bug. Another example is that all users do not always use an application software program with a bug. In these cases, only some portion of terminals exhibit unusual behaviors. Monitoring all session creation request messages results in a small change of the reconnecting terminal ratio, i.e., x_m.

Thus it is necessary to group mobile terminals which are affected by the same bug and then to calculate a change of the reconnecting terminal ratios for the groups of mobile terminals. However, it is not clear how mobile terminals are

grouped. In Sect. 4, we will study empirically what kinds of the groups are useful by analyzing 6 month log data of session creation request messages.

4 Detection examples of software failures

We analyzed the 6 month log data of session creation request messages in a mobile data communication system. The system which was targeted by our log data analysis handled over 1.5 million mobile terminals and over 100 mobile terminal models. A part of the mobile terminals were connected to the system at the same time, and the mobile data communication services were requested randomly. We successfully detected anomalies caused by failures of session management nodes and a radio access network. The details are described in [2]. In this section, we show how a bug of an application software program by a third-party application provider on a mobile terminal is detected by our algorithm. The bug makes a mobile terminal re-send a session creation request message in some conditions. This section describes how such a bug is detected using our anomaly detection technique.

4.1 How k for threshold t_m is determined

To detect anomalies it is important how k for threshold t_m in Equation(6) is determined. If k is set to a large value, some anomalies may be missed. On the contrary, if k is set to a small value, many fake anomalies which are not caused by failures are erroneously detected. The anomaly detection tool is an operations tool and thus anomalies are reported as "*alarms*" to operators. It is important not to report many fake alarms. We set a goal that the number of fake alarms is less than 0.2 percent. Since one day consists of 480 sample periods, about one fake alarm would be reported in average per day.

To determine such k, we investigated the relationship between g_m and t_m with various values of k. Fig.5 shows one-day result of g_m and t_m with k=1.5, 2, 2.5, 3, 3.5 and 4 from the log data. The curves in Fig.5 correspond to the thresholds t_m at all sample periods with k=1.5, 2, 2.5, 3, 3.5 and 4. Each circle g_m in Equation(6) corresponds to the difference between the reconnecting terminal ratio and its exponential average at each sample period. If a circle is over the curve, it is an anomaly. By counting the number of such circles for the 6 month log data, we choose 3 as k such that anomalies are detected at about one percent of total sample periods.

4.2 How the bug is detected based on reconnections

We apply our algorithm into the 6 month log data. We calculate the reconnecting terminal ratio x_m for all mobile terminals in the system. The anomaly detection detected anomalies as shown in Fig.6(C). It shows the time-series of g_m (calculated from x_m by Equation(4)) and the curve of threshold t_m with k=3 during

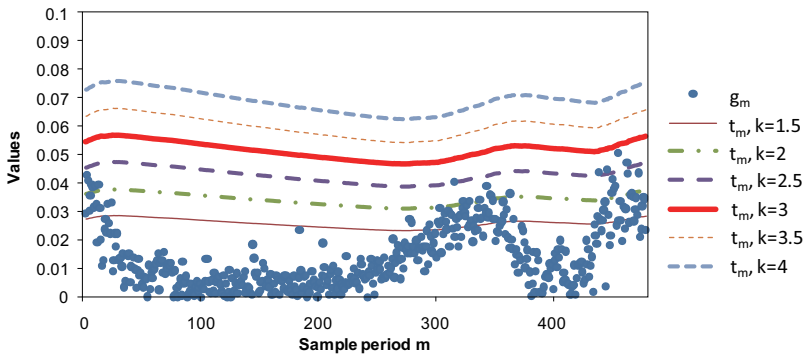

Fig. 5. One-day result of g_m and t_m with various k

40 sample periods. At the sample periods 4 and 24 in Fig.6, values of g_m are over the threshold t_m and these points are detected as anomalies.

However, it was not unknown which component's failure caused these anomalies because there were many candidates of failures causing these anomalies. Although we checked logs of session management nodes, application servers and equipments in the radio access and core networks, no failure was found. Thus we suspected that mobile terminals would be affected by a bug of an application software program on mobile terminals. At this time, since we heard a new release provided by a third-party vendor for mobile terminals, we assumed that this release might have a bug. We validated this assumption in the following steps: First, we found candidates of mobile terminals which might have downloaded this release. Please note that some of these candidates have not downloaded it yet. Second, mobile terminals are divided into two groups: (*Group A*) the group of such candidate mobile terminals, and (*Group B*) the group of other mobile terminals.

Figure 6(A) and (B) show the results for these two groups. Although anomalies are detected in *Group A*, no anomaly is detected in *Group B*. As the result, we consider that this anomaly would be affected by this release and then actually found the bug in this release of application software program.

This grouping was useful to identify the bug, but this fact implies that the anomaly could be detected earlier if we monitored only mobile terminals of this group. In this empirical study, some days passed since the new release of the software program had been announced. It means that before starting to collect the log data, the release was announced and many mobile terminals already downloaded it. If we monitored this group of mobile terminals, this anomaly would be detected earlier. We consider that the monitoring for anomaly detection per such groups of mobile terminals is important to quickly detect it.

To validate the above hypothesis, since we do not have the log data before the new release date, we investigated the relationship between g_m and t_m with k=3 for all mobile terminals and the group of them with the bugs (*Group A* and

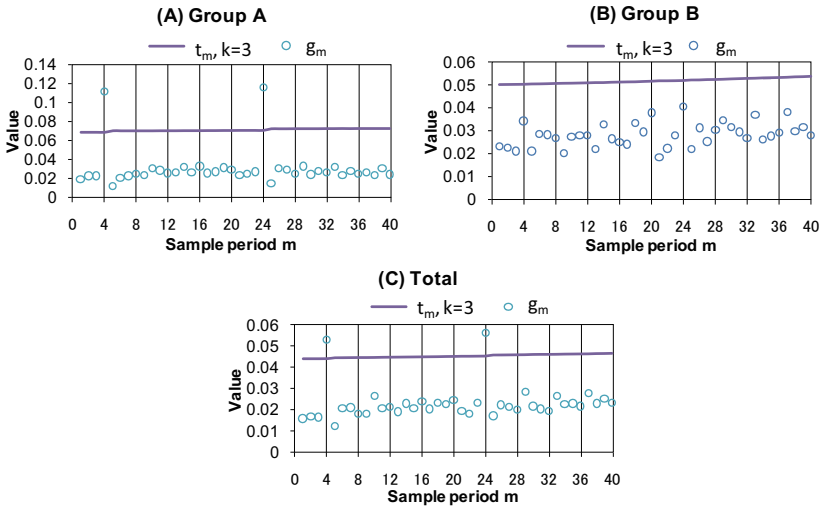

Fig. 6. Time-series on g_m and t_m with k=3

Total is the same as Fig.6) after several days of releasing the patch (the program of fixing the bug) for the software program. Our assumption is that the number of mobile terminals with the bug was decreased since some mobile terminals downloaded the patch to fix the bug and that in this case, the anomaly is detected in *Group A*, but it is not detected in *Total*. If this assumption is correct, we can detect such an anomaly by focusing on a group of mobile terminals with a bug. Figure 7 shows the graphs of the relationship after several days of releasing the patch. Since the circle as g_m over the curve as t_m with k=3, anomalies occur at the sample period of 8 and 24 in *Group A* of Fig.7(A). On the contrary, in *Total* of Fig.7(B), there is no anomaly.

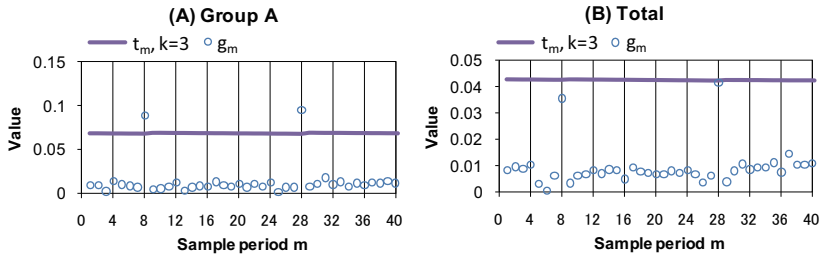

Fig. 7. Relationship between g_m and t_m with k=3 after several days of releasing the patch

5 Related work

There are two approaches for anomaly detection in communication systems [1]: traffic monitoring and probing approaches. Our anomaly detection tool takes the traffic monitoring approach. In this section, we discuss about the major related work [1].

5.1 Traffic monitoring

Traffic monitoring is a passive scheme whereby messages (packets) are monitored (observed) to detect anomalies.

Statistics-based monitoring
Statistics-based monitoring means that a management node collects the statistics on packet transmission from nodes such as layer2 switches and routers by communicating with the nodes (e.g., [5]). Such communication overheads between the management node and the other nodes are not negligible. Since the system consists of a number of nodes, i.e., application servers, session management nodes and so on, the overhead would be too heady and time-consuming.

Packet capturing-based monitoring
Packet capturing-based monitoring means that some equipment captures transmitted packets and obtains packet transmission statistics. However, since a huge number of packets (dozens of gigabits per day) are transmitted on a commercial mobile core network, this method has a few disadvantages: Many capturing points are needed to collect transmitted packets. Besides, the method of trajectory sampling [6] can decrease the number of packets that need to be captured. However, such a sampling would be prone to miss anomalies.

On the contrary, our tool only captures session creation request messages (packets) which contain user's requested services. This means that the number of captured packets less than the number of all transmitted packets. All session creation request messages can be obtained at a few (maybe one) capture points with a few PCs.

5.2 Probing

Probing is an active scheme whereby probes which check whether equipments are not in failure are sent to the equipments.

Internal probing
Probe programs are installed into equipments such as communication nodes or application servers, and they check whether the equipments are not in failure [7]. However, such probe programs are difficult to install if the service nodes are not in networks administrated by operators. In our system, they cannot be installed on third-party application providers' servers.

External probing
A probe tool checks equipments by actually sending test messages to servers [8, 9]. This external probing is not scalable for a large-scale network. Many hours would be needed to check all components such as application servers, session management nodes and mobile terminals.

6 Conclusion

In this paper, we applied an anomaly detection technique to detect anomalies caused by mobile terminal software failures (bugs). The anomaly detection technique focuses on user's behavior to reconnect a service and it detects anomalies based on how many mobile terminals re-send session creation request messages. Although this is a light-weight mechanism, it enables to quickly (within a few minutes) detect anomalies caused by not only communication failures, but also bugs. This empirical study shows that a bug of mobile terminal was actually detected. We consider that anomaly detection techniques are useful to detect bugs which were overlooked during a testing period. As far as we know, this paper is one of the first papers which actually detected a bug in a commercial environment. Although this paper does not deal with software testing techniques, an anomaly detection technique is complementary to these techniques and plays an important role to detect bugs as soon as possible.

In the future, we try a remaining issue about how to find such a group of mobile terminals which are affected by the same bug. We also consider how to apply our anomaly detection algorithm to other systems. Since our algorithm can be applied to session-request based system, for example, we may apply it easily to the IP Multimedia Subsystem (IMS) [10].

References

1. Marina T. and Chuanyi J.: Anomaly Detection in IP Networks. IEEE Transactions on signal processing, Vol. 51, No. 8, pp. 2191–2204. (2003)
2. Hiroyuki S., Satoshi K., Hideyuki K., Teruyuki H. and Hidetoshi Y.: A scheme for detecting communication abnormality in mobile core networks based on user behavior. The 12th International Symposium on Wireless Personal Multimedia Communications (WPMC 2009). (2009)
3. Sumaru N.: Experimental Study on User Behavior Analysis by Keylogs of Cellular Phone (in Japanese). IEICE 5th Workshop on Brain Communication. (2008)
4. NIST/SEMATECH: e-Handbook of Statistical Methods. http://www.itl.nist.gov/div898/handbook/, Sect.6.4.3.1. (2006)
5. Case J., Fedor M., Schoffstall M. and Davin J.: Simple Network Management Protocol (SNMP). RFC1157. (1990)
6. Duffield N.G, Gerber A. and Grossglauser M.: Trajectory Engine: A Backend for Trajectory Sampling. IEEE Network Operations and Management Symposium 2002. (2002)
7. Matthias W., Peter U. and Xavier D.: An SNMP based failure detection service. The Proceedings of 25th IEEE Symposium on Reliable Distributed Systems (SRDS'06). (2006)

8. Irina R., Mark B., Natalia O., Sheng M. and Genadv G.: Real-time Problem Determination in Distributed Systems using Active Probing. IEEE/IFIP Network Operations and Management Symposium, 2004 (NOMS 2004), Vol.1, pp. 133-146. (2004)
9. Wolfgang B.: Nagios, Second Edition, System and Network Monitoring. No Starch Press. (2008)
10. Gonzalo C. and Miguel-Angel G.: The 3G IP Multimedia Subsystem: Merging the Internet and the Cellular Worlds. Wiley (2004)