New Method for Software Updating in Mobile Phones

R. Kiyohara[*] and S. Mii[**]

[*]Mitsubishi Electric Corporation
5-1-1, Ofuna, Kamakura-Shi, Kanagawa, 247-8501, Japan
[*]Information Technology R & D Center, [**]Nagoya Works
{Kiyohara.Ryozo@ah, Mii.Satoshi@dx}.MitsubishiElectric.co.jp

ABSTRACT

Due to the increase in cellular phone services (e.g., i-mode), car navigation systems, and other embedded devices, it is nowadays difficult to release devices that are bug-free. Therefore, there is a requirement for fixing bugs after the shipment of devices to the end-user with over-the-air (OTA) updating of device software. In addition, new software functions are required to be installed on the devices after shipment. OTA updating and new software installations take place repeatedly; this involves managing a considerable amount of binary difference (delta), which is difficult. In this paper, we propose a new broadcasting model for software updating and a new method to merge two delta versions (e.g., the delta between versions 1.00 and 1.1 and the delta between versions 1.1 and 1.2) into one delta version (e.g., the delta between versions 1.0 and 1.2) for mobile devices. This paper evaluates and presents the results of our new broadcasting model and software updating method.

Keywords: Mobile Phone, Software Updating, Binary Difference, Software Management, Delta

1 INTRODUCTION

In recent times, there has been a rapid increasing in the size of the software on mobile phones equipped with a wide range of features—such as an internet browser, email client, camera, infrared connection, Java virtual machine, and local wireless functions. Complex processes, such as event handlers, require that bugs that are fixed after the shipment of a particular product are fixed through services like MS Windows Update to ensure that personal information is protected from loss or leakage.

A large number of mobile phone manufacturers or carriers provide software update services such as over-the-air (OTA) updates or through distribution centers[1][2][3]. The time required for these update services is very crucial. During OTA updates, naive users may not recognize the software updating mechanism, and may remove the battery when the mobile phones are not being used by them. During software updates through distribution centers, the time required for an update determines the capacity of the service, because the updating capacity of a distribution center is limited.

The updating time comprises the following two phases;

(1) Downloading data for updating purpose.
In several cases, binary delta techniques have been adopted to update from old versions to new [4] [5]. Such techniques can provide the data required for small software updates.

(2) Updating the flash memories on target devices.
Mobile phone program code is stored on flash memories after the resolution of address references as execute-in-place (XIP) code. In a large number of cases, the software structure significantly influences the size of the rewritten codes.

Because it is difficult to fix all bugs at the same time, these updating services are repeatedly used by mobile devices,. Moreover it is wise to assume that new bugs can be inserted. Therefore, bug-fixing data should be broadcasted, though the user should decide whether to update or not.

In this paper, we propose a new delta technique for software updating services which require repeated release of new software versions for the same mobile devices. Furthermore, we evaluated our technique, and it yielded good results.

2 SOFTWARE UPDATE

2.1 Software Update for Mobile Phone

Figure 1 shows a typical software update system for mobile phones. In the development environment, a bug-fixed software image is created and the delta data are generated through binary delta technologies. Such systems

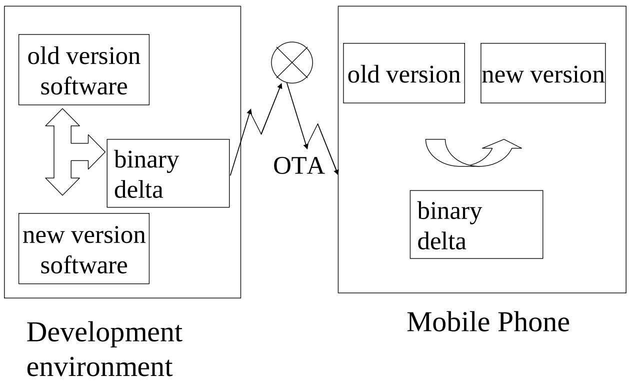


Figure 1: Software Update for the Mobile Phones

face the following challenges;

(1) Downloaded data size should be as small as possible.
To decrease the data size when rewriting the program code for new versions, binary difference (called delta) techniques are required [6]. Generally, the program code for a mobile phone is stored in XIP format, which is executed directly without using dynamic links. This feature increases the delta size. Hence, a binary delta technique for program code has to be adopted to reduce the size of the delta. We present some binary delta techniques in Section 3. During the download phase, a user is able to use a large number of functions except those functions that require a large amount of memory (e.g., camera, Java applications).

(2) The time required for rewriting new software on a mobile phone should be as small as possible
Flash memories cannot be erased as individual pages, but as large blocks. Figure 2 demonstrates a problem in which a large amount of code is changed despite adding only a single line; this is important for software structures. During the rewriting phase, users cannot access any function, which is similar to the BIOS updating function in PCs. From our experience, the long interval of time during which users cannot access the services of their mobile phones causes them to try to reset their phones, similar to rebooting a PC. However, resetting during software updating destroys the program code.

2.2 Proposed OTA Service Model

Figure 2 shows our model for OTA services that repeatedly release new software versions. When a new version is released, the delta data between new and old versions are released. Currently, many services release some delta data between the newest versions and each software version in the mobile phone which end-user has; that is, there are three kinds of delta which are the delta between R2.00 and R.1.20, the delta between R2.00 and R1.10 and the delta between R2.00 and R1.00 in the Figure 2. The software updating services are executed only when the device requests it. As the Software update is requested at the same time just after new version has been released, software updating services have to be scheduled for each mobile phone or require a wait time.

Therefore, to overcome this problem, delta data should be broadcasted to only those mobile devices that have not received the delta. Moreover, software downloading and rewriting of the flash memories should be distinguishable by the user. Mobile devices receive the delta data, and the user decides whether to update or not. In this proposed model, the carrier or manufacturer cannot manage the software versions for each devices. Therefore, deltas between the new version and all old versions have to be broadcasted. However, this results in a lot of traffic on the cellular network and should be avoided.

Therefore, we propose a new software updating method to broadcast only one kind of delta data between the newest and previous software versions.

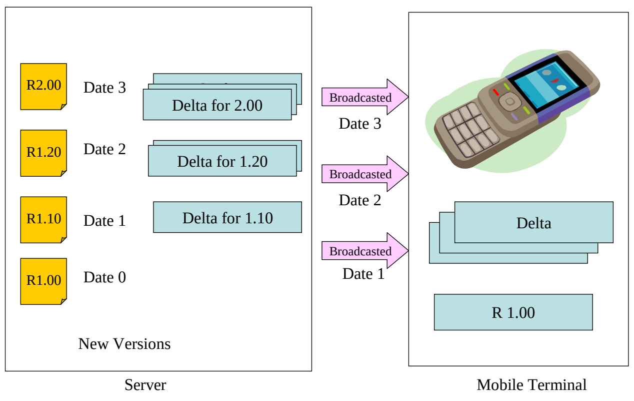


Figure 2: OTA Service Model

3 RELATED STUDIES

Currently, OTA update services can be used to reduce the maintenance cost of the software on mobile phones [1]. Some OTA solutions have been proposed and implemented for this purpose [2][3]. In such services, it is important to develop technologies to achieve the following two issues;

(1) Reduction of the binary differential data between old and new versions of program code to reduce the amount of data downloaded OTA, and reduction of the delta data for improving usability and reducing the maintenance cost.

(2) Reduction of the number of pages written to flash memories to reduce the time required for rewriting them. This is because during the rewriting phase, users are unable to access any other function. In most cases, the mobile phone does not have extra flash memories; therefore, software update functions rewrite the program codes directly while suspending the services.

A large number of studies on binary delta have been based on using the diff [7] algorithm on UNIX operating systems [8]. In the mobile computing field, these techniques are applied to disconnected operations and synchronization for file systems [9][10] in narrow band network environments. Previous studies [4][5] have successfully contributed to the development of technology for this software updating with regard to binary images. These techniques focused on the rules for register assignment or the address part of the instruction code to deduce the delta size.

The time required for the rewriting phase depends on the software structure. A large number of reference parts in the instruction code can be changed. Moreover, when NAND flash memories are used and their program code is compressed, the rewriting time depends on the compression time. In [11], attempts were made to solve this problem by introducing a new compression algorithm.

However, these studies focused only on the size of the delta or the rewriting time for a single time. There have been no studies for repeatedly downloading or rewriting updates for a mobile phone with limited memory and bandwidth.

4 DELTA

4.1 Delta Format

In this section, we introduce the format for the binary delta used in this paper. The format is based on the gdiff [12] or VCDIFF [13] format. These are the standard formats for Internet users. Figures 3 and 4 shows the basic concept and an example of the format.

In Figure 3, area 1 is the same between both versions. Areas 2 and 4 are the same except for the stored address. These areas are represented in the delta as COPY commands. The COPY command requires the original start address and length of the area.

Area 5 is added. Added data are represented as a data command. A DATA command is a one-byte command, and the command itself shows the length (e.g., DATA command "10" means that the following 10 bytes are the new data). Therefore, the DATA command only requires the new data.

Area 3 is deleted. However, the deleting command is not represented because this area is rewritten by other commands.

The size of the delta depends on the number of commands. Let c denote the number of COPY commands, d denote the number of DATA commands, and l denote the average length of new data. Let the address and length information be 4 bytes. The COPY command requires the old address information and the length. The size of delta is then

$$Delta = 9c + ld \qquad (1)$$

4.2 Updating Models

For the OTA updating model described in Section 2.2, the delta data between the newest update and the last previous version are broadcasted every time a new version is released. However, the software of a device is not always the previous last version. If the software is an older version with a different delta, the device cannot be updated because the software on the device is not consistent with the delta.

Therefore, there are two updating models:

(1) Store the all deltas on the device. When a user wants to update, the delta data are applied step by

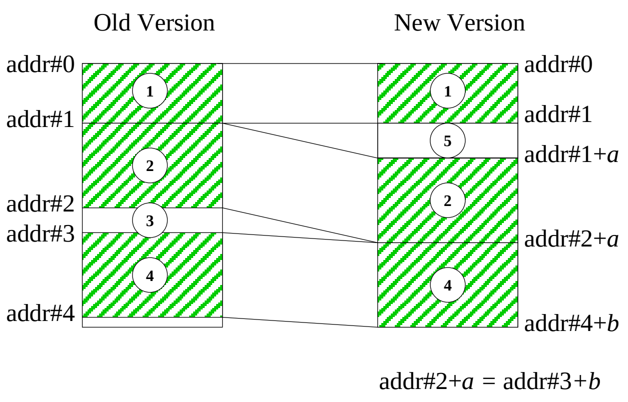


Figure 3: Binary difference

Delta :COPY, addr#0,(addr#1-addr#0)
DATA, *a*, *xxxxxx*
COPY,addr#1,(addr#2-addr#1)
COPY,addr#3, (addr#4-addr#3)

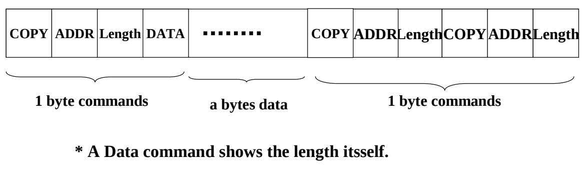


Figure 4: Format of the delta command

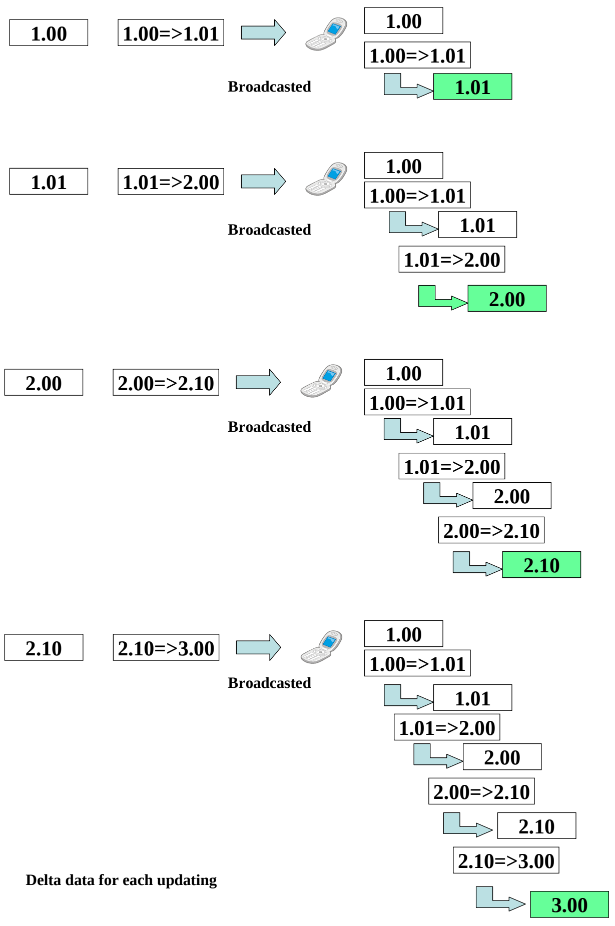


Figure 5: Model for storing all delta data.

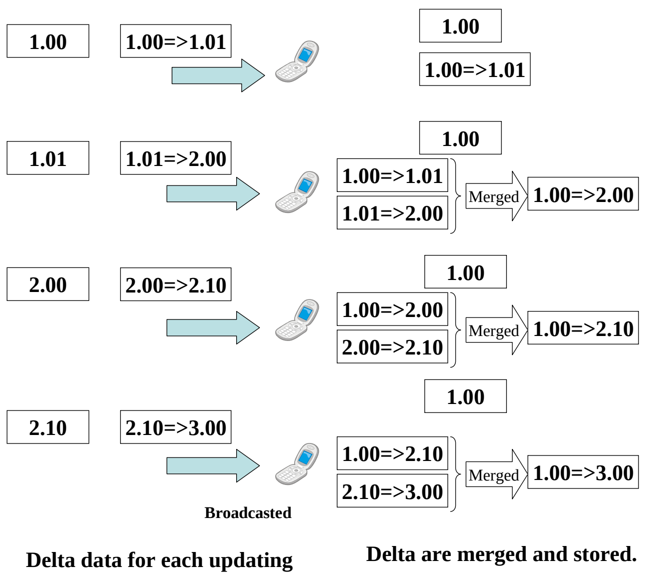


Figure 6 Model for storing the merged delta

step (see Figure 5).

(2) Store the merged delta on the device. When the delta is received, the new delta is merged with old delta (see Figure 6).

In (1), the device has to keep each delta. Therefore, the storage for updating required the amount of delta data. This storage limits the user storage for many data (e.g., e-mails, photos, Java applications, etc.). Moreover, updating step by step makes the flash memories rewrite repeatedly. Therefore, the updating process requires a long time during which the user cannot use the mobile phone. However, the benefit is that implementation is simple.

In (2), the device has to keep the delta for the newest version and the software on the devices. The delta is merged on the device when the new delta is received. The storage requires only one version of the delta data. However, the merged process is complicated despite the limited resources on the device. Therefore, we propose a new method for merging the delta data with limited resources.

5 PROPOSED METHOD

5.1 Basic Algorithm

We propose a new method for merging the delta on a device with limited resources. Figure 7 shows the example of the delta for three versions. The left side is the delta between versions 1.0 and 2.0. The right is the delta between versions 2.0 and 3.0. Figure 8 shows the merged delta. The first COPY command in the delta from version 2.0 to 3.0 means to copy the code from version 2.0. However, the software version on the device is version 1.0. Therefore, the command is divided into two. One is the COPY command, and the other is the DATA command from the delta between versions 1.0 and 2.0.

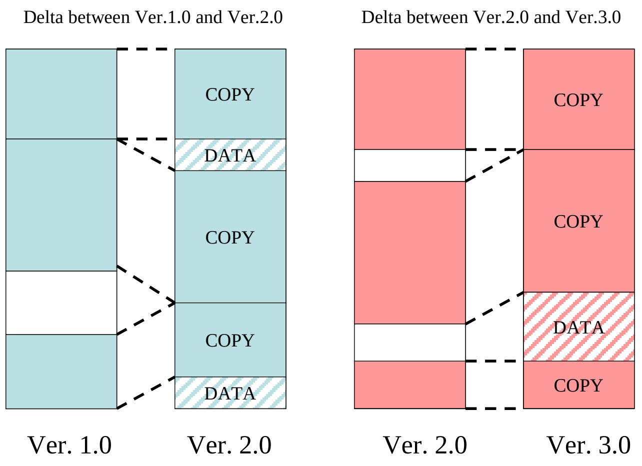


Figure 7: Example of delta for three versions

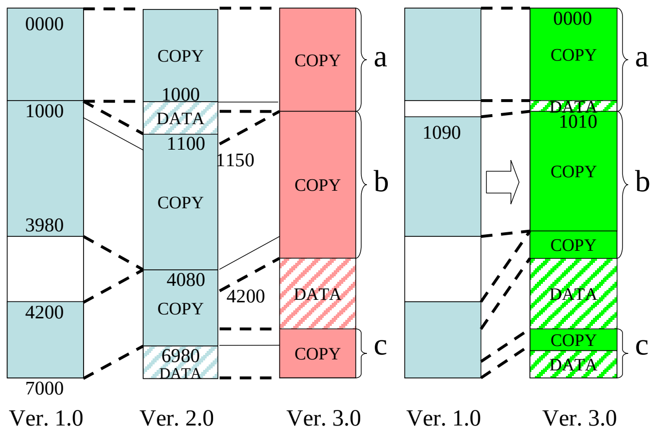


Figure 8 : Example of merging the delta

All commands in the delta between versions 2.0 and 3.0 should be checked and merged. The delta is merged by the following steps:

(1) The delta data are analyzed from the top of the new delta data, —i.e., analyzed from the low address to the high address.

(2) DATA commands should not be changed.

(3) COPY commands have to be decided using COPY or DATA commands by searching the old delta data by address information—i.e., the command checks whether the code exists on the current version or not.

In the example shown in Figure 7, the merged delta is shown on the right side of Figure 8 and the delta commands are shown in the Figure 9.

There are often many changes in the address part of instruction code. Therefore, in this checking phase, there are many search processes. Thus, this process should be scalable for a device with limited resources.

```
Ver. 1.0 => Ver. 2.0
   COPY     0, 1000
   DATA    100,  xxxxxx
   COPY    1000, 2980
   COPY    4200, 2800
   DATA    20,  xxxxxx

Ver. 2.0 => Ver 3.0
   COPY    0, 1010
   COPY    1150, 3050
   DATA    2000, xxxxx
   COPY    6060, 940

Merged delta  Ver. 1.0 => Ver. 3.0
   COPY    0,1000
   DATA    10, xxxx
   COPY    1050, 2830
   COPY    4200, 120
   DATA    ……
   COPY
   DATA
```

Figure 9: Example of delta commands

5.2 Index Table

The delta data are commonly analyzed from the low address portion and applied. Therefore, it does not have to search the delta data by the address information and there are no indexes. However, this method requires indexes. If there are indexes for the address information, the searching process is executed in *O(log n)*. However, the data size of the delta might be very large. In some case, it might be larger than the delta itself. The size depends on the number of commands. The address information of the new and old versions and the size information are required for each command. Figure 10 shows an example of the index table. It shows that 12 bytes data are added when 4 bytes are being addressed. From equation (1) , the delta size is

$$Delta = 21c + ld \qquad (2)$$

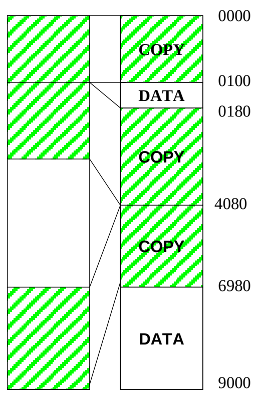
However, the storage which the proposed method requires is only 2.3 times of deltas in the worst case. This is because the COPY commands require the 12 bytes information (two address information and size information).

6 EVALUATION

In service models, usability depends on the size of the storage required on the mobile devices, and the time for rewriting the flash memories. During the rewriting phase, users cannot use the mobile phones, as in Windows BIOS updates. Therefore, it should be evaluated with this in mind.

Moreover, from the point of view for the cellular network, the data size of broadcasted should be evaluated.

For the evaluation, we compared the total size of broadcasting data and the required storage on the mobile devices for the following three methods:



	Address of new version	Size of new version
1	0	100
2	100	80
3	180	3900
4	4080	2900
5	6980	2020

Index table

Figure 10: Example of index table

Table 1: Software size of the mobile phones

Mobile phone name	The Software Size
I	45.1MB
II	21.4MB

# Data Sections are not included

Table 2: Size of the delta in mobile Phone I

Version name	2.0	3.0	4.0
1.0	127,452	128,118	155,254
2.0		714	76,171
3.0			75,498

Bytes

Table 3: Size of the delta in mobile Phone II

Version name	2.0	3.0	4.0
1.0	202,111	532,187	542,793
2.0		708,015	718,578
3.0			92.014

Bytes

(a) Legacy models. The delta data for each version are broadcasted.
(b) The delta data for recent versions are broadcasted and updating step by step.
(c) The delta data for recent versions are broadcasted and merged the delta on the devices.

6.1 Measurement

We measured these kinds of data for two types of 3G mobile phone software running on an ARM processor. Table 1 shows the software size of the mobile phones. Software updating is only for bug fixation. For each mobile phone, we measured the delta information using four versions (1.0, 2.0, 3.0, and 4.0).

Tables 2 and 3 show the size of the delta between each version. There are various delta data. For example, the delta size between versions 2.0 and 3.0 in mobile phone I is a very small pattern. The size of the delta between versions 2.0 and 3.0 in mobile phone II is a very large pattern.

Table 4: Size of the merged delta in mobile phone I

Version name	3.0	4.0
1.0	128,126	158,840
2.0		76,172

Bytes

Table 5: Size of the merged delta in mobile phone II

Version name	3.0	4.0
1.0	624,428	639,206
2.0		726,832

Bytes

Table 6: Evaluation of the merged delta size in mobile phone I

Pattern	Merged delta	Amount of delta	Direct delta
1.0=>3.0	128,126	128,166	128,118
1.0=>4.0	158,840	203,664	155,254
2.0=>4.0	76,172	76,212	76,171

Bytes

Table 7: Evaluation of the merged delta size in mobile phone II

Pattern	Merged delta	Amount of delta	Direct delta
1.0=>3.0	624,428	910,126	532,187
1.0=>4.0	639,206	1,002,140	542,793
2.0=>4.0	726,832	800,029	718,578

Bytes

6.2 Size of Merged Delta

Tables 4 and 5 show the size of merged delta data. The merged deltas are expected to be larger than the direct deltas between each version and smaller than the total sum of the deltas.

The results are shown in Tables 6 and 7. In all cases, the sizes of the merged delta data were smaller than the total amount of delta data and a little larger than the sizes of the direct delta data. However, the ratio depended on the updating patterns. For example, the size of the delta from version 2.0 to 3.0 in mobile phone I is very small. Therefore, the size of merged delta was not so different from the size of the total or direct delta.

However, in mobile phone II, the size of the merged delta was much smaller than the total delta and only 20% larger than the direct delta. These results were as expected.

6.3 Size of Delta Data to be Stored

Method (a) requires the storage for only recent delta data, because the delta data for each version are always broadcasted. Method (b) requires the total amount of delta data because if a user attempts to update the software version, the updating process is executed step by step. Method (c) requires the recent delta data, which are merged with previous delta data and the size of the index table. This index table might be stored twice.

In Tables 8 and 9 compare these data. For this case, method (c) is better than method (b). In the worst case, method (b) is better than method (c) because the data do not include the size of indexes. However, this index size

Table 8: Required storage in mobile phone I

Method	Size of required storage(Bytes)		
	1.0=>3.0	1.0=>4.0	2.0=>4.0
(a)	128,118	155,254	76,171
(b)	128,166	203,664	76,212
(c)*	128,126	158,840	76,172

* not include the index table

Table 9: Required storage in mobile phone II

Method	Size of required storage(Bytes)		
	1.0=>3.0	1.0=>4.0	2.0=>4.0
(a)	532,187	542,793	718,582
(b)	910,126	1,002,140	800,029
(c)*	624,428	639,208	726,832

* not include the index table

Table 10: Size of delta data to be broadcasted in mobile phone I

Method	1.0=> 3.0	1.0=> 4.0	2.0=>4.0
(a)	255,570	410,824	76,885
(b)	128,166	203,664	76,212
(c)	128,166	203,664	76,212

Bytes

Table 11: Size of delta data to be broadcasted in mobile phone II

Method	1.0=> 3.0	1.0=> 4.0	2.0=>4.0
(a)	255,570	410,824	76,885
(b)	128,166	203,664	76,212
(c)	128,166	203,664	76,212

Bytes

depends only on the number of delta commands and does not depend on the software versions. Therefore, the size of the delta that needs to be stored on the device does not have to be taken care of in these three methods.

6.4 Rewriting Time

Most of the rewriting time depends on the number of blocks which have to be erased from the flash memories. methods (a) and (c) erase the flash memories to be rewritten only once. However, method (b) has to erase the number of steps. In many case, the same blocks need to be rewritten.

Method (a) and the proposed method (c) are obviously fast. Recently, this rewriting time has been very long. Some models in the Japanese market take more than 20 min to rewrite flash memories. Even though there are few changes in the program code, there are many changes in the binary code because sliding the positions of the program code changes the many reference pointers.

Moreover, during the rewriting phase, it is impossible to use the mobile phone. Therefore, rewriting time is very important for mobile phone users.

6.5 Size of Delta Data to be Broadcasted

Tables 10 and 11 show the total data size to be broadcasted for each method. Methods (b) and (c) have the same data size because both methods require only the delta between the newest and the previous versions. Method (a) requires the delta between the newest version and the current version on the devices. Therefore, broadcasting is not suitable. In these tables, method (a) shows the download data size. Actually, since many current versions are expected, the delta data for all versions are broadcasted and the mobile devices selected the delta for their version.

Based on the data size to be broadcasted, methods (b) and (c) are effective.

7 CONCLUSION

We proposed an OTA service model that broadcasts only one kind delta data between the newest and previous versions of mobile phone software. The model separates the downloading phase (i.e., broadcasting phase) from the rewriting phase. Users can decide whether to update or not.

In this model, it is difficult to decide which delta should be broadcasted. We found that the mobile devices can store the delta data. Therefore, all delta should be applied to the flash memories when users need to update. However, storage on mobile devices is limited and rewriting time should be small.

Therefore, we proposed a method for merging the delta. We compared the proposed method with the method of updating in multiple steps. Moreover, we also compared the simple method, wherein the delta for all versions is broadcasted.

All methods were the equivalent in terms of required storage, which depends on the data. The problem faced by the proposed method was with regard to the size of the index.

The rewriting time of the proposed method was the same as the simple method. The rewriting time for the method for updating in multiple steps was a serious issue.

The data size to be broadcasted for the proposed method was the same as that for the method for updating in multiple steps. However, the data size in the simple method was a serious issue.

Therefore we can conclude that our proposed method can be effectively applied to our proposed model.

In future work, we will modify this merged method to develop a more complicated delta format. Further, we will attempt to apply this technique to the other types of software updating.

REFERENCES

[1] Hoshi S., Ichinose A., Nose Y, Hosokawa A, Takeichi M., and Yano E., “Software Update System Using Wireless Communication,” NTT DoCoMo Technical Journal, Vol.5, No.4, (2004), 36–43.

[2] Innopath, “Understanding Firmware over the Air-FOTA,” http://www.innopath.com/pdf/fota.pdf.

[3] Red Bend Software, “Mobile Software Management Solutions,” http://www.redbend.com/solutions/index.asp.

[4] Kiyohara R., Kurihara M.,Mii, S., and Kino S., “A Delta Representation Scheme for Updating between Versions of Mobile Phone Software,” Electronics and Communications in Japan, Vol.90, No.7, (2007), 26–37.

[5] Terazono K. and Okada Y., “An Extended Delta Compression Algorithm and the Recovery of Failed Updating in Embedded Systems,” Proc. IEEE Data Compression Conference 2004, (2004), 571.

[6] Takeichi M.,Hosokawa A.,Nasu K. ,Hoshi S, Moriyama K, Takami T, and Terunuma K:Bug fix of mobile terminal

software using download OTA, The Asian-Pacific Network Operations and Manage-ment Symposium(2003) 6.3

[7] Hunt W. G. and Gzymanski G. T., “A Fast Algorithm for Computing Longest Common Subsequences,” CACM, Vol20, No.5, (1997), 350–353.

[8] Tichy F. W., “The string-to-string correction problem with block moves,” ACM Trans. Computer systems， Vol.2, No.4, (1983), 309–321.

[9] Balasubramaniam S. and Pierce B., “What is a File Synchronaizer?,” ACM MobiCom ’98, (1998), 98–108.

[10] Tridgell A. and Mackerras P., “The rsync algorithm,” Australian National University, TR–CS–96–05 (1996).

[11] Kiyohara R. , Mii S., Matsumoto M., Numao M. and Kurihara S., “new method of fast compression of program code for OTA updates in consumer devices,” IEEE Transaction on Consumer Electronics,(2009), 812-817

[12] Hoff v. A. and Payne J., “Generic Diff Format Specification,” http://www.w3.org/TR/NOTE-gdiff-19970901

[13] Korn D., MacDonald J., Mogul J. and Vo K., “The VCDIFF Generic Differencing and Compression Data Format,” RFC 3284, http://tools.ietf.org/html/rfc3284