Vol.2011-CDS-1 No.1
2011/6/15

コンシューマデバイス向けソフトウエア更新方式

清原良三[†1]　寺島美昭[†1]　田中功一[†1]

携帯電話やカーナビゲーションシステム，さらにWeb-TVなどのコンシューマ機器のソフトウェア規模が巨大化しつつあり，最近では不具合なしの状態での出荷が困難な状況である．また，OSなど基本機能の強化開発も継続的に行われることが多く，ソフトウェアの機能のアップグレードへの要求も高い．そのため，ソフトウェアの更新をネットワーク経由で行うサービスがある．一方，同じプラットフォーム上でソフトウェアでも，基本的なＯＳやライブラリのアップグレードで動作保証がないといったことが十分考えられ，ユーザは自分で選択的にソフトウェア更新を行う必要性が出てくると想定できる．このような様々なケースを想定すると，ソフトウェアのアップグレード情報を個々のデバイスからサーバに取得要求をする場合，転送要領や更新時間を考慮した差分更新という手法では，バージョン管理の手間が大きくなり現実的ではない．つまり最新版のプログラムやデータを取得するとそのデータ量は差分といってもデータ量は大きなものになると考えられる．有線で接続されている場合は，Windows Updateサービスのようにデータサイズは問題にならないかもしれないが，広域の携帯電話網を使うような場合にはできる限りデータサイズは小さい方が良い．そこで，本論文ではこのような課題を解決するためのソフトウェア更新モデルを提案した上で，そのモデルを実現する上で必須となる世代間差分情報合成方式を提案し，その評価を行い有効性を示す．

Merged Delta Method for S/W updating on Consumer Devices

Ryozo Kiyohara,[†1] Yoshiaki TerashimaISL
and Koichi TanakaISL

Due to increasing the size of software for cellular phones, car navigation systems, Web-TV and other consumer devices, it is difficult to release bug-free devices. Therefore,there is a requirement for fixing bugs after the shipment of devices and over the air (OTA) updating of device software like the Windows update services. On the other hand, there is a requirement of installing the new software functions for the devices after shipment. These kinds of updating make software updating repeatedly. Therefore, it is difficult to manage a lot of delta data. In this paper, we proposed the model of software updating and the method of converting the two delta for one delta for consumer devices. we have also evaluated it.

1. はじめに

携帯端末やカーナビゲーションシステム，Web-TV等のソフトウェア規模が巨大化し，最近では不具合なしでの出荷が困難な状況である．また，新機能の追加の要求などもあり，OSやドライバなどの下位レイヤーのソフトウェアのアップグレードが繰り返し行われることを想定する必要が出てきている．

ソフトウェアの不具合修正や機能のアップグレードを繰り返し行う場合，多数あるデバイスはどのバージョンがどのの状態になってるか管理する手間が大変であり，すべてのバージョンから最新のバージョンにバージョンアップする仕組みが必要になる．そのため，サーバ上でのデータ管理などが複雑になる[1)2)3)]．

また，コンシューマデバイスでは，ユーザが勝手にインストールしたソフトウェアが動作していることもあり，複雑に機器のライブラリを使いこなすケースを想定しておくべきである．あらゆる機能の動作が確認できるまではバージョンアップしたくないケースもある．例えばPC用のブラウザなどは典型的な例でバージョンアップにより動作しなくなるアプリケーションが存在する．

そのため，配布された更新データは保持し，いつでもバージョンアップできるようにはしておくが，実際には必要になるまでバージョンアップしないということもある．とくに，最近のスマートフォンのようにユーザが自由にアプリケーションをダウンロードできるようになると特定のライブラリのバージョン時のみ動作するということも十分考慮する必要がある．

このようなことを考慮すると，現状の最新版に必ずバージョンアップするモデルではデータ転送量も多くなり，また管理すべきバージョン間差分情報なども増加し，現実的でなくなる．

本論文では，サーバ上のデータ管理を単純化するために，複数世代間に渡るバージョンアップに関して，各世代間の差分を合成することにより，ソフトウェアの書き換えによる性能の劣化を防ぐ方式を提案し，転送すべきデータサイズに関して評価し，効果があることを示す．

†1 三菱電機（株）
MITSUBISHI ELECRIC CORPORATION

2. 関連研究

ソフトウェアの更新に関しては様々な研究がある．例えばシステムとしてはネットワークで繋がれた環境でデータを送って適用更新するための研究[4]がある．また，世界最大のこの種のサービスと言って良いマイクロソフト社の Windows Update サービスに関する研究[5]もある．また，ソフトウェアはライセンスの問題もあるため，その観点からの研究[6]もある．しかしこれらは，システムに関する研究であり，常に最新のバージョンに更新することしか想定しておらず，複数のバージョンで様々な要求がある場合を想定できていない．そのため，そのため試験不足でリリースして不都合が生じる場合も少なくない．

また，不具合の修正といった小さいな変更よりむしろ機能の追加といった要求が多く，不具合の修正のようなマイナーなバージョンアップも含む場合に対してはそのままの考え方を適用することはできない．

ソフトウェアを配布する上では旧版と新版のソフトウェアの差分を抽出し，差分を送ることがネットワークトラフィックの観点からも有効な方式である[7][8]．あるいは，オブジェクトの変更履歴からソフトウェアの差分更新を行う方式[9]などの研究もあり，直接バイナリの差分を取ることに比べて，アドレス情報などのずれを意識せずにすむことで，場合によっては有効な手法になる．また派生バージョンが発生した場合のプログラム差分に関する研究[10]もある．また，差分そのサイズそのものを小さくするために，一部をシンボルレベルで差分を抽出する[13]技術の開発も行われ，ブラウザのバージョンアップなどにも適用されつつある．

一方コンシューマ機器ではフラッシュメモリを使う場合が多く，フラッシュメモリの書換えは時間のかかる場合が多い．特にソフトウェアの更新中は機能を使えなくなる場合が多く，ユーザは何が起こっているかわからずリセットを試みる可能性も高い．そのため高速な書換えが要求される．ソフトウェア更新時のフラッシュメモリの書換えを意識した研究もある[?]が，複数世代に渡る書換えまでは考慮されていない．

しかしながら，コンシューマ機器のように必ずしも広帯域のネットワークに繋がるという保証がない場合において，世代が複数にまたがった場合に繰り返し更新があり，しかも様々なバージョンが存在する場合の多世代に渡る効率的な更新方法に関しては有効な手法は存在していない．

3. 差分更新

3.1 ユーザモデル

想定するサービス，ユーザのモデルを図 1および以下に示す．

(1) 出荷したコンシューマ機器はソフトウェアのバージョンアップを複数回に渡り実施．
このバージョンアップには不具合の修正，機能の更新，他ライブラリとのバージョン整合性を要求する場合などがある．

(2) バージョンアップのためのデータは携帯電話網などの帯域の狭いネットワークを経由して配布する場合も想定．
コンシューマ機器は世界中で使われることを想定する必要があり，環境の良いところのみを想定しないが，少なくともネットワークには接続可能な状態をここでは想定する．ネットワークに接続不能な場合には，媒体経由での配布でも更新可能であると考えるが，本論文ではネットワーク接続の場合に関して述べることとする．

(3) データ量が少なくする差分データを機器に対して送付．
転送時間の問題即ちコストに係わる問題と，書換え時間（最近の携帯電話の更新では数十分かかるケースすらある）の問題を考慮し，バイナリ差分での配布とする．

(4) ソフトウェアの配布は一斉同時配信させ，一斉配信できなかったデバイスは個別に要求．
個別にデバイスから要求するモデルでは，サーバへの負荷が一時に集中するという問題があり，携帯電話などでは時間の予約を行うような手法をとってアクセスの集中を防いでいる．しかし，台数の多いコンシューマ機器でしかも要求はいつ発生するかわからないようなモデルではデータの管理の手間も大きいため，一旦データは一斉同時配信する方が良いと考える

(5) ユーザは受信したデータを自デバイスに保存するものの，本当にソフトウェアをバージョンアップするかどうかはユーザ自身が不具合の内容や更新された機能と，バージョンアップのリスクとを考えてユーザがバージョンアップするかどうかを選択
バージョンアップにより動作しないアプリケーションが発生するようなケースに関しては，ユーザの責任でバージョンアップすることが望ましい．

Vol.2011-CDS-1 No.1
2011/6/15

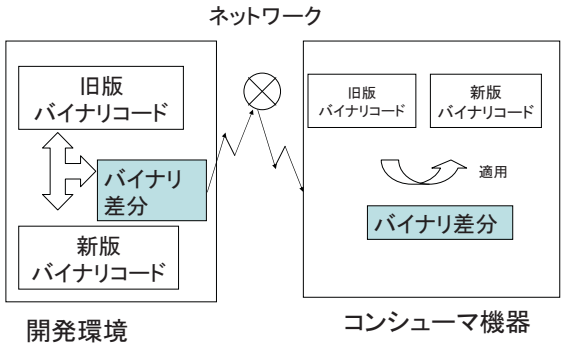

図 1 ソフトウェアバージョンアップモデル

```
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)
```

図 2 gdiff フォーマット

3.2 差分更新の原理

差分更新の一般的な原理を説明する．バイナリレベルの差分は多くの場合，図3に示すようなCOPY コマンドと DATA コマンドで差分データを表現し，図2に示すフォーマットでバイナリ表現する[14)15)]．COPY コマンドは旧データのアドレスと新データのアドレス情報を持ち，デバイス上でコピーを行う．さらに携帯電話など組込み系のデバイスではアドレス解決済みのプログラムコードが置かれているためにリファレンス先が移動するとその影響を受ける部分が多くなり，差分が大きくなる．そのため移動量のデフォルト値を導入し，実際にはデータが異なっても差分として出さずに，デバイス上で計算して置き換える方式[7)] がある．移動によって代わった参照部分をケアする必要がなくなり差分は小さく表現できる．一般に差分を小さく表現するための工夫は多くの場合，アドレスのずれやレジスタアサインのずれを補正するという方式で実現している．

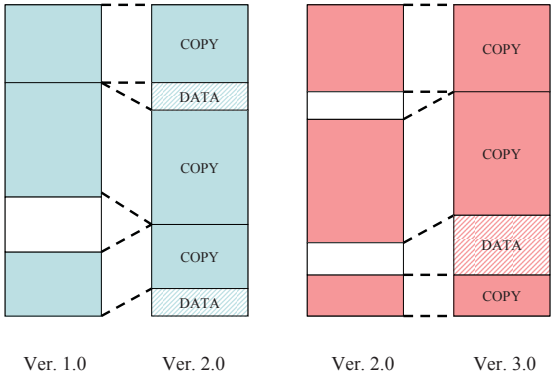

図 3 差分表現方式

COPY コマンドの数を c, 各 1 バイトの DATA コマンドとそのデータの長さを l バイトとし，データコマンドの数を d とする．また，1 バイトで表す COPY コマンドごとにアドレス情報として 2 つ分合計 8 バイトの情報が必要だとすると，一般に差分量は以下の式 1で示される．

$$Diff(v1, v2) = 9c(v1, v2) + \sum_{i=0}^{d-1} l_i(v1, v2) \quad (1)$$

3.3 差分更新モデル

前節での述べたユーザモデルに対して，差分更新によるサービスモデルは図5,6, 7という 3 つのバージョンアップ方式が考えられる．

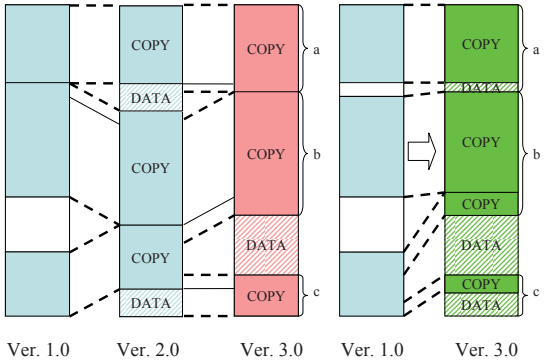

図 4 単純な差分合成

(1) 全バージョン管理モデル

図 5に示すように，網羅的に更新データを用意しておき，配布時にはデバイスのバージョンに応じたデータを送付するというサービスがまずは考えられる．しかしながら，この方式では図 5のバージョン 1.00 の状態で， 1.01 へバージョンアップするための差分データを受け取りながら，実際にバージョンアップをしない場合では，バージョン 2.00 にバージョンアップする際には， 1.01 にバージョンアップするためのデータを破棄し，バージョン 2.00 へアップするためのデータをダウンロードする必要が出てくる．つまり，前回ダウンロードしたデータが無駄になる．また，このモデルでは， 2.10 まで最新版がなっているときに，途中の 2.00 版までバージョンアップしようとしてもできない．もし，そのようなバージョンアップをこの方式で実現しようとすると，すべのバージョン間の差分データを用意する必要があり，$O(n^2)$ のデータ数をサーバ上に用意する必要があり現実的ではない．

(2) 順次バージョンアップモデル

図 6に示すように連続するバージョン間の差分のみをデバイス上におき，デバイスは順番に何度もバージョンアップするという手法が考えられる．この方式では以前に取得した差分データは無駄にはならないが，必要なすべての差分データを保持しておく必要がある．デバイスのメモリは有限であり望ましくない．また，フラッシュメモリを利用している場合には，何度も書き換えながら更新していくことになり，デバイスの書き換え時間が深刻な問題となる．例えば，最近の携帯電話のバージョンアップで数十分かかるケースがあり，これを繰り返すと相当な時間がかかることが十分予想できる．

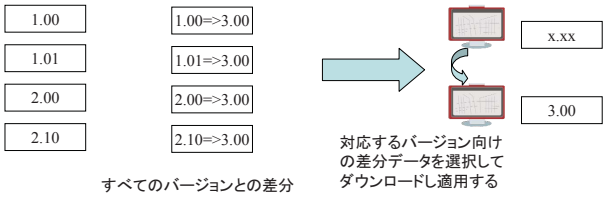

図 5 全バージョン管理方式

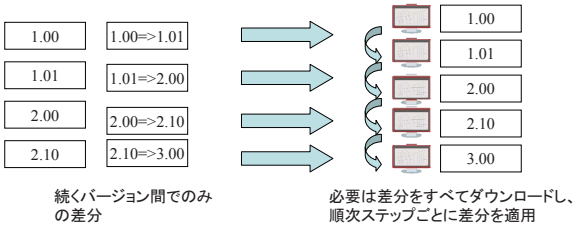

図 6 順次バージョンアップ方式

4. 提 案 方 式

4.1 提案更新モデル

データの管理は，連続するバージョン間の差分のみとした上で，図 7に示すようにバージョンアップに必要なデバイスの要求に応じて動的に複数の差分の情報から 1 つの差分情報に合成する手法が考えられる．この手法では，新しい差分データを受信完了後，直ちにデバイス上に保存していた差分データと受信したデータを合成した新しい差分データを作成し，デバイス上に保持する差分データを 1 種類のみとする．このようにすることにより，更新時のフラッシュメモリの書き換えも 1 回で済む．しかし，差分データは，書き換え対象のデータが存在していることを前提に作成してあるため，単純には合成することはできない．

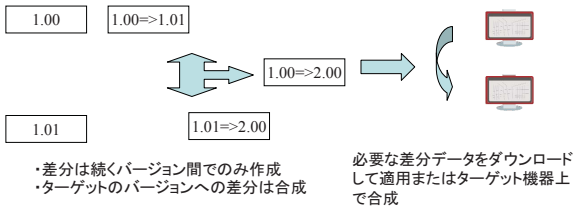

図 7 提案方式

4.2 提案差分合成方式

提案する手法は，図 7に示すように，差分を合成することによって，送信データを一つにまとめ，かつ小さく表現可能とする方式である．

以下に示す方針で，差分の合成を行う．

(1) 新しいバージョンへの差分情報を先頭から解析する．

(2) DATA コマンドの場合はそのまま出力する．

(3) COPY コマンドの場合は，旧版のアドレスを元に古い方の差分情報を参照し，旧版の情報としてデバイス側に存在する情報かどうかを判定して，COPY コマンドとデータコマンドを使い分けて出力する．

図 3の例で，Ver.1 から Ver.3 に更新する場合は，差分合成は図 4に示すようにようになる．例えば，a の部分の COPY コマンドは古い差分コードの一部が COPY になっているものの，一部は DATA コマンドである．このような場合は，DATA コマンド部分が必要になるため，a は，一部 COPY と DATA に分かれる．同様に b の部分は複数の異なる部分からの COPY コマンドに分かれる．c もまた，COPY コマンドと DATA コマンドに分かれる．

CPU の貧弱なデバイス上でのこれらを実行するが，検索処理はテーブルでデータを管理しておけば，2 分探索で COPY コマンドにすべきか DATA コマンドにすべきかを探すことができ，平均処理時間はコマンド数 n に対して $O(\log n)$ で処理することができるためあまり問題にならない．差分コマンド数がある程度大きくなっても，実行時間はフラッシュメモリへの書き込み時間に比べると無視できるほど小さくできる．

また，高速にこれらのテーブルを検索するために，テーブル上のデータは差分コマンドごとに，新旧双方のアドレス情報，サイズ情報が必要となり，4 バイトアドレッシングの場合で 1 コマンドあたり 12 バイト余分に必要になる．これは，差分情報に比較してテーブルサイズの方が大きくなる可能性があるが，2 バージョンの差分情報を置く程度のことであり，複数世代に渡る更新を考えると無視して良いと考える．

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

図 8 合成前差分コマンド例

Merged delta Ver. 1.0 → Ver. 3.0

COPY	0,1000
DATA	10, xxxx
COPY	1050, 2830
COPY	4200, 120
DATA	……
COPY	
DATA	

図 9 合成後差分コマンド例

5. 評　　価

本提案方式の評価のポイントは以下に示す 4 点である．

(1) デバイス上に保持する差分データサイズの大きさおよび数
(2) データの準備時間（差分抽出時間)

(3) 要求が来てからのデータ準備時間
(4) ダウンロード時間

デバイス上に保持しなければならない差分データの許容できる大きさはメモリの空き容量に依存してくる．そのため一定量までは許容範囲になるが，一定量を超えると問題となる．つまり，世代数に比例することなく一定量であることが望ましい．

データ準備時間はサーバ上での操作時間であり，試験をしてから公開することと，サーバマシンのスペックに依存することから気にする必要がないが，スケーラビリティがないと手間が増加するため，定性的評価は必要であろう．

要求が来てからのデータ準備時間は，差分合成方式の合成時間の問題となるが，ダウンロード時間および書き換え時間に比べ小さく，ダウンロードのみして書き換えない場合はバックグラウンドでも動かすことができるため無視できる．

ダウンロード時間はユーザの操作性に大きく関係する項目であるが，ダウンロードデータサイズに比例するため，ダウンロードデータサイズで評価する．これはサービス側からも網への影響を考えると重要なファクターである．

これらを以下の３つの方式で比較評価する．

(a) 毎回直接最新版に更新する方式
(b) 世代ごと順に更新する方式
(c) 差分合成方式

5.1 評価対象データ

評価にはサイズの大きい携帯電話のソフトウェアイメージを利用した．対象データの特性を表１ , 表 2, 表 3に示す．評価は 3G 携帯電話の 2 機種の出荷時ごろの不具合修正データを利用した．表 1には，データ部分をソフトウェア更新の対象となるコードの数と評価に使ったバージョン数を示した．

また，表 2, 表 3では，バージョンアップの場合のみを対象とし，各世代間の差分の大きさを示した．評価の観点から様々な場合を考え，差分量が小さいもの，大きいものを評価対象としてそれぞれ選んでいる．また，一部には書き換えた内容を誤った修正を元に戻すような特徴のあるコードも含めている．

表 1 評価対象ソフトウェア

データ種別	データサイズ 単位 (M バイト)	使用バージョン数
携帯電話機種 1	45.1	4
携帯電話機種 2	21.4	4

上記データサイズは評価対象とした部分のサイズ

表 2 評価対象機種 1 差分情報

旧版＼新版	B(バイト)	C(バイト)	D(バイト)
A	127,452	128,118	155,254
B		714	76,171
C			75,498

表 3 評価対象機種 2 差分情報

旧版＼新版	B(バイト)	C(バイト)	D(バイト)
A	202,111	532,187	542,793
B		708,015	718,578
C			92,014

5.2 差分合成測定結果

差分データの合成によって，差分サイズがどのようになるかを表 4および表 5に示す．また，表 6, 表 7にこれらの合成差分が，差分の合計および直接差分抽出した場合との比較した結果を示す．

合成差分サイズは機種 1 の場合は，とくに A → C などではほとんど変わらない．これは B → C の差分がほとんどないからである．このように差分のほとんどない場合は，差分合成してもデータサイズはほとんど変わらない．逆に機種 B のような場合には，差分合成すると各世代の差分を合計するよりは明らかに小さくなる．しかしながらダイレクトに差分をとるよりも差分は大きくなるが，3 世代に渡る程度では 20% 程度の増加に抑えられている．

5.2.1 ダウンロードデータサイズ

ユーザにソフトウェアをブロードキャストするあるいは，自動的に取得させるというモデルでは新しいソフトウェアがリリースされるたびにデータをダウンロードすることになる．ただし，

表 4 評価対象機種 1 合成差分サイズ

旧版＼新版	C(バイト)	D(バイト)
A	128,126	158,840
B		76,172

表 5 評価対象機種 2 合成差分サイズ

旧版＼新版	C(バイト)	D(バイト)
A	624,428	639,206
B		726,832

表 6 評価対象機種 1 差分比較

パターン	合成差分サイズ (バイト)	各差分合計 (バイト)	直接差分 (バイト)
A → C	128,126	128,166	128,118
A → D	158,840	203,664	155,254
B → D	76,172	76,212	76,171

表 7 評価対象機種 2 差分比較

パターン	合成差分サイズ (バイト)	各差分合計 (バイト)	直接差分 (バイト)
A → C	624,428	910,126	532,187
A → D	639,206	1,002,140	542,793
B → D	726.832	800.029	718,578

適用するかどうかはユーザの意思による．この場合，(b),(c) 方式では総データダウンロード量は，最新の版を $newest$ とした場合以下の式で表される．

$$TotalDownloadSize = \sum_{v1=0}^{v1=newest-1} Diff(v1, v1+1) \quad (2)$$

つまり毎回配布されるデータの総和である．これを蓄積していくことになる．一方 (a) の方式では毎回最新版へのバージョンアップのための差分情報を要求することになるため，1 世代の差分ではない情報が送られてくることになる．

評価対象コードでは，1 度も実際のソフトウェアを書き換えを行っていないと仮定した場合に，表 8, 表 9に示す結果となった．ダウンロードデータサイズは機種 1 ではかなりの削減が達

表 8 機種 1 ダウンロードデータサイズ比較

方式	ダウンロードデータ合計サイズ (バイト)
(a)	410,824
(b)	203,664
(c)	203,664

表 9 機種 2 ダウンロードデータサイズ比較

方式	ダウンロードデータ合計サイズ (バイト)
(a)	1,277,091
(b)	1,002,140
(c)	1,002,140

成できる．機種 2 でも 20% は削減できている．機種による削減率の差で悪くなるケースは，元のコードに戻ったり，例えば，レジスタ退避シーケンスなどの偶然ではあるが元と同じプログラムコード部分が同じになってしまう場合などは，版の離れたバージョン間での差分がそれほど大きくならないため，削減率が減少する場合があると考えられる．

5.3 保持データサイズ

デバイスに保存する必要のあるデータサイズは，デバイスに保持できるユーザデータの制限サイズと大きくかかわるため，できるだけ小さい方が良い．方式 (a) では毎回最新版に更新するため差分データは最新のもののみ保存しておけば良い．しかし方式 (b) では逐次バージョンアップを行うためすべての差分を保存する必要がある．提案方式である (c) では差分情報を逐次合成していくため，1 世代前の差分情報のみを保存しておけば良いことになる．

それぞれを評価データを用いて測定した結果を表 10，表 11に示す．当然のことながら，方式 (a) が最良の結果となる．方式 (b) は最悪の結果となるが，方式 (c) は機種 1 ではあまり変わらず，機種 2 ではちょうど中間的な値となっている．いずれにしろ，すべてを蓄積する方式に比べると提案方式ははるかに良いことがわかる．

5.4 結　　論

差分データ合成によるソフトウェア更新方式の提案方式を実際の 3 G 携帯電話のプログラムコードを用いて評価した結果，提案方式で，1 世代分の差分データを順次適用する場合と比較して，デバイス上に保持すべき差分データサイズは 3 世代の差分で 50% 削減できることがわかった．また，常に最新版に書き換える方式と比べてもダウンロード合計時間が短くなり，網への影響を考慮すると十分に効果があることがわかった．

表 10 機種 1 保持データサイズ比較

方式	保持でデータサイズ最大値
(a)	155,254 バイト
(b)	203,664 バイト
(c)	158,840 バイト

表 11 機種 2 保持データサイズ比較

方式	ダウンロードデータ合計サイズ
(a)	542,793 バイト
(b)	1,002,140 バイト
(c)	726,832 バイト

しかし，効果が薄くなるケースもあり，一度変更したコードを元に戻すような修正が入った場合に冗長な差分コマンドが多数発生し，効果が薄くなっていると考えられる．

6. おわりに

提案したソフトウェア更新方式で，ダウンロードデータ量およびデバイス上での保持データ量の削減に一定の効果があることがわかった．しかしながら効果が薄い場合もある．その原因は冗長な差分コマンドによるところが大きいと考えられる．

今後，この冗長さを押さえ，かつデバイス上での計算をなるべくしないで済むような方式を検討するとととともに，より最適かしたいくつかの差分表現方式においても効果があることを検証していく予定である．

参 考 文 献

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, pp.36-43,(2004)
2) Innopath: Understanding Firmware over the Air-FOTA (online), available from ⟨ http://www.innopath.com/pdf/fota.pdf ⟩ (accessed 2011-05-07).
3) Red Bend Software: Mobile Software Management Solutions (online), available from ⟨ http://www.redbend.com/solutions/index.asp ⟩ (accessed 2011-05-07).
4) Hemmerich C.: Automatic request-based software distribution, USENIX 14th System Administration Conference,pp.197-206(2000).
5) Dunagan J., Roussev R., Daniels B., Johnson A., Verbowski C., and Wang M. Y. : Towards a self-managing software patching process using black-box persistent-state manifests, IEEE Int. Conf. on Autonomic Computing,pp.106-113(2004)
6) Sobr L. and Tuma P.:SOFAnet Middleware for software distribution over Internet, In IEEE Symp. on Applications and the Internet (SAINT ' 05)(2005).
7) 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, pp.26-37(2007).
8) 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,p.571(2004).
9) 寺島美昭，別所雄三，宮内直人，中川路哲男，鹿間敏弘，福岡久雄，佐藤文明，水野忠則, "差分更新を実現する分散オブジェクト再構成ミドルウェアの実装と検証,"情報処理学会論文誌, Vol.46, No.9 pp.2288-2299
10) 栗原まり子, 清原良三, 橘高大造，渡辺拓, 古嶋寛之, "インターネットを利用した大規模ソフトウェアのバージョンアップ方式に関する検討," FIT2004,M-010
11) Gkantsidis C., Karagiannis T. and Vojnovic M.: Planet Scale Software Updates, Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for Computer Communications,pp.423-434(2006).
12) Chromium:Software Updates: Courgette (online) available from ⟨ http://dev.chromium.org/developers/design-documents/software-updates-courgette ⟩ (accessed 2011-05-07)
13) 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, Vol.55, No.2, pp.812-817(2009)
14) Hoff A. and Payne J.:Generic Diff Format Specification (online), available from ⟨ http://www.w3.org/TR/NOTE-gdiff-19970901 ⟩ (accessed 2011-05-07)
15) Korn D., MacDonald J., Mogul J. and Vo K.: The VCDIFF Generic Differencing and Compression Data Format, RFC 3284 (online): available from ⟨ http://tools.ietf.org/html/rfc3284 ⟩ (accessed 2011-05-07)