Vol.2011-SLDM-149 No.23
Vol.2011-EMB-20 No.23
2011/3/18

放送による車載機器向けソフトウェア差分更新方式

施　欣漢[†1]　中西恒夫[†1,†2]
久住憲嗣[†2]　福田晃[†2]

車載機器に搭載されるソフトウェアの大規模化，複雑化が進むにつれ，ソフトウェアの欠陥を除去しきらないまま製品を出荷する危険が増しつつある。本稿では，放送局において新旧のファームウェアをバイナリレベルで比較して生成した差分ファイルを放送し，車載機器において当該差分ファイルに基づいて古いファームウェアを新しいものに更新するソフトウェア差分更新方式を提案する。提案手法では，車載機器における厳しいメモリ制約，ならびに不安定な電源環境の問題に対処すべく，既存のバイナリファイル差分更新ツールである `bsdiff/bsupdate` に改造を加える。具体的には，新しいバイナリ保管用のメモリを必要としないインプレース型差分更新を実現し，コンテキストを定期的に保存することで更新処理の不意の中断からの回復を可能にする。

Differential Update of Automotive Device Firmwares with Broadcasting

Hsin-Han Shih,[†1] Tsuneo Nakanishi,[†1,†2]
Kenji Hisazumi[†2] and Akira Fukuda[†2]

As automotive device firmware becomes larger and more complicated, the risk to release products with defects is increasing. This paper proposes a software update scheme that uses broadcasting, *i.e.* the station broadcasts a differential between old and new firmwares and the automotive device updates its own old firmware to the new one based on the differential. The proposed scheme reforms `bsdiff/bsupdate`, existing tools for binary file updating, to resolve problems specific to automotive devices such as memory constraint and unstable power supply. We achieve in-place differential updating which does not require memory for the new binary and enable safe recovery from sudden power cuts by storing contexts periodically.

1. はじめに

今日，自動車に搭載されているコンピュータ，ECU（Electronic Control Unit）の数は1台あたり数十を数える。ECUに搭載されるソフトウェアも大規模化，複雑化の一途をたどっており，いまや自動車全体においてソースコード行数で1,000万行に及ぶ規模になっている[1)]。当然，ソフトウェアに起因する不具合も増える傾向にあり，出荷前に欠陥の混入を防ぐ形式手法や欠陥の除去を図るテストの技術はもちろん，出荷後に欠陥を除去するソフトウェア更新技術も今後重要になってくるであろう。

現在，自動車の制御に直接関わる，ECUに搭載されるソフトウェアの更新は専ら整備工場において行われている。また，カーナビゲーションシステムなど自動車の制御に直接関わらないソフトウェアの更新は，DVD/CDやSDメモリカード等のメディアを介したユーザによる更新が行われている。整備工場でソフトウェアの更新を行う場合，つまりリコールを行う場合，自動車メーカには相当のコストが発生する。ディーラがユーザから自動車を受け取って整備工場まで運び，ソフトウェアを更新し，ユーザに自動車を返しに行くまでのコストに加え，ユーザへの通知や整備工場で車を預かっている間の代車等のコストも発生する。一方，ユーザ自身でソフトウェアの更新を行う場合はこうしたコストはかからないが，ユーザには臆することなく機器を操作してソフトウェアを更新することが問われる。

ソフトウェアの出荷後の自動更新に関して，PCではインターネットを介して製造者が事前に用意したサーバにアクセスして，インストール済みソフトウェアのバージョンが最新のものかを確認し，必要に応じて最新版のソフトウェアをダウンロード，インストールすることが一般的に行われている。組込みシステムについては，たとえば携帯電話では移動体通信網を用いたPC同様のファームウェア更新手法が，テレビではデジタル放送を用いたファームウェア更新がすでに実用化されている。

本稿では放送を用いた車載機器向けのファームウェアの更新手法を提案する。メモリ容量の制約，不安定な電源環境，広くはない放送の帯域，不安定な受信環境など車載機器ゆえの問題から，PCや携帯電話，家電等で用いられているソフトウェア更新手法を，そのまま車載機器に対して適用するのは困難である。本稿では，放送局が新旧ファームウェアの差分情

†1 九州大学大学院統合新領域学府
Graduate School of Integrated Frontier Science, Kyushu University
†2 九州大学大学院システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu University

Vol.2011-SLDM-149 No.23
Vol.2011-EMB-20 No.23
2011/3/18

報を放送し，車載機器は受信した差分情報に基づいて自身の持つファームウェアを新しいものに更新する手法を提案する。新ファームウェア全体を送るのではなく，新旧ファームウェアの差分情報を送ることにより，放送波の帯域使用量と車載機器における受信バッファ用の要求メモリ量を下げる。また，旧ファームウェアを直接書き換えるインプレース型更新を実現することで更新時のメモリ消費量をさらに抑える。あわせて，エンジン停止等によりファームウェア更新中に給電が止まっても，給電再開後にファームウェア更新に正しく復帰させる更新手順について述べる。

本稿第 2 節ではソフトウェアの差分更新に関する関連研究を紹介する。本稿では，既存のバイナリファイル差分抽出／適用ツールである `bsdiff/bsupdate` を車載機器向けに改造する。第 3 節で `bsdiff/bsupdate` について紹介し，第 4 節で `bsdiff/bsupdate` を車載機器向けに改造する。最後に第 5 節でまとめを述べる。

2. 関 連 研 究

ソフトウェアの更新方式には，サービスを停止してソフトウェア更新を行うオフライン更新方式，サービスの提供を継続しつつソフトウェア更新を行うオンライン更新方式[2] に大別される。本稿では，ソフトウェア更新サブシステムの簡略化のためにオフライン更新方式[3),4)] を検討する。

ソフトウェアの更新方式は更新を行う粒度の観点でも分類できる。サブシステム単位での更新方式，モジュール単位[5)] での更新方式，バイナリレベルでの更新方式などが考えられる。車載機器では多様なプロセッサが使用されている。あらゆるプロセッサにおいてモジュール単位での更新を可能にするためには，それら多様なプロセッサを一元的に抽象化する必要が出てくる。本稿では，やはり簡略化のため，特定のプロセッサを前提とせず，また更新内容の意味解釈を必要としないバイナリレベルの更新方式を検討する。

PC 上で使用されるバイナリファイル差分抽出ツールはいくつも実装されている。

`bdiff`[6)] は旧バイナリファイルの接尾辞木を構築し，その接尾辞木を用いて新旧バイナリファイルの最長共通文字列を探索し，新旧バイナリファイルの差分を抽出する。`bdiff` では新旧バイナリファイルの差分を旧バイナリファイルに対する編集指令の列として表現する。編集指令は，旧バイナリの指定位置から指定バイト数のバイト列を複写する指令と，旧バイナリファイルに対するバイト列を追加する指令の 2 種類がある。

`Vdiff/Vdelta`[6)] も `bdiff` 同様，接尾辞木を用いた最長共通文字列探索アルゴリズムで新旧バイナリファイルの差分を抽出する。差分を旧バイナリファイルに対する編集指令の列として表現するのも同様であるが，編集指令の構成が若干異なる。また，`bdiff` が差分ファイルを圧縮しないの対して，`Vdiff/Vdelta` は LZ77 による差分ファイルの圧縮を行う。

`Zdelta`[6),7)] は，旧バイナリファイルの接尾辞木ではなく接尾辞配列を構築し，その接尾辞配列をハッシュ探索することで新旧バイナリファイルの差分を抽出する。差分はやはり旧バイナリファイルに対する編集指令の列として表現されるが，差分ファイルを小さくすべく，複写指令をより多く生成するようにしている。また，差分ファイルを拡張ハフマン木を用いたアルゴリズムで圧縮する。

このほかに本稿で採用している `bsdiff`[8)] がある。`bsdiff` は実行形式ファイルに対して，他のツールより相当に小さな差分ファイルを生成できることが知られている。

3. バイナリファイル差分抽出／適用ツール： bsdiff/bsupdate

本稿では，PC 上でしばしば利用されているバイナリファイル差分抽出／適用ツールである `bsdiff/bsupdate`[8)] を車載機器向けに改造し，放送によるファームウェア更新を実現することを検討する。`bsdiff` は新旧異なるファイルのバイナリファイルの差分を抽出するツール，`bsupdate` は `bsdiff` により生成された差分を古いバイナリファイルに適用して新しいバイナリファイルに更新するツールである。本節では，`bsdiff`，`bsupdate` について概説したい。

`bsdiff` は新旧のバイナリファイルを比較し，図 1 に示す構成の差分ファイルを生成する。新旧のバイナリファイルを比較した場合，新旧のバイナリファイルは，古いバイナリファイルから新しいバイナリに複写されるバイト列，古いバイナリファイルから削除されるバイト列，新しいバイナリファイルに新規に追加されるバイト列に分割することができる。`bsdiff` は，接尾辞配列を応用したアルゴリズムを用いて新旧のバイナリファイルを比較し，古いバイナリファイルを新しいバイナリファイルに更新するための編集命令列を生成し，差分ファイルの編集命令領域に格納する。ひとつの編集命令はバイト数の三つ組 (κ, ν, σ) で表現される。`bsupdate` は，このように表される編集命令を，編集命令領域の最初から順番にひとつずつ読み出し，以下の処理を行う。但し，更新開始時に古いバイナリファイルの現在参照点はファイルの先頭に設定されているものとする。

(1) Copy & Modify：古いバイナリファイルの現在参照点から κ バイトのバイト列を読み出し，新しいバイナリファイルの末尾に追加する。さらに，差分ファイルの差分データ領域から κ バイトのバイト列を読み出し，新しいバイナリファイルの末尾に追加されたバイト列に加算する。古いバイナリファイルの現在参照点は κ バイト分，

Vol.2011-SLDM-149 No.23
Vol.2011-EMB-20 No.23
2011/3/18

末尾方向に進める。

(2) New：差分ファイルの追加データ領域から ν バイトのバイト列を読み出し，新しいバイナリファイルの末尾に追加する。古いバイナリファイルの現在参照点は移動しない。

(3) Skip：古いバイナリファイルを σ バイト分読み飛ばす。すなわち，新しいバイナリファイルの末尾には何も追加せず，古いバイナリファイルの現在参照点を σ バイト分，末尾方向に進める。

bsdiff，bspatch の現在の実装では，差分ファイルの編集命令領域，差分データ領域，追加データ領域には bzip2 による圧縮が施され，差分ファイルの一層の小型化が図られている。もっとも差分データ領域，追加データ領域の圧縮には bzip2 以外のアルゴリズムも原理上使用可能である。

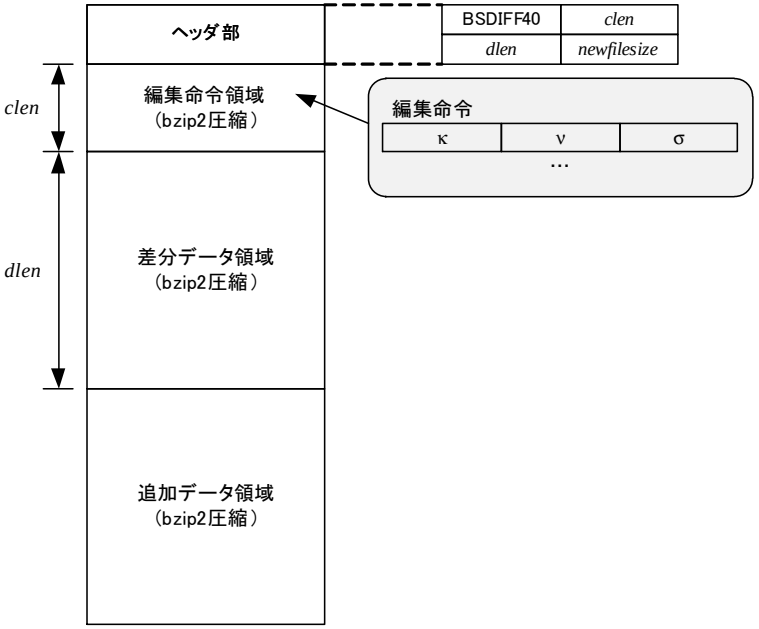

図 1 bsdiff/bspatch 差分ファイルの構成

4. bsdiff/bspatch の車載機器向け改造

放送局は bsdiff で生成した新旧バイナリの差分ファイルを放送し，車載機器は放送された差分ファイルを受信し bspatch により旧バイナリを更新することで，車載機器ファームウェアの放送による差分更新を実現する。しかしながら，bsdiff，bspatch は専ら PC 上で利用されるバイナリファイル差分抽出／適用ツールであり，車載機器向けに特別の考慮はなされていない。本節では，車載機器向けバイナリファイル差分適用ツールに求められる要件を述べ，当該要件を満足すべく bsdiff/bspatch を改造する。

4.1 車載機器における制約

放送を用いて車載機器のファームウェアを更新する場合，以下の問題について検討する必要がある。

- メモリ容量の制約：量産される車載機器は製造コストの削減は収益を左右する。一般に車載機器は PC と異なり，あらかじめ決まった機能しか提供しない，ひいてはあらかじめ決まったソフトウェアしか実行しない。そのため，ハードウェア資源，特に RAM は必要最小限の容量しか搭載されない。
- 不安定な電源：家庭内で用いられる家電機器と異なり，車載機器に給電が行われるのは，エンジンがかかっているか，エンジンキーがアクセサリ（ACC）ポジションにある間のみである。ファームウェア更新中であってもユーザはエンジンを切るかもしれない。
- 不安定な受信環境：上に述べた通り，車載機器が差分ファイルの放送を受信できるのは，エンジンがかかっているか，エンジンキーが ACC ポジションにある間に限られる。自動車が動いている時間は 1 日平均 1〜2 時間程度と言われている。また，トンネルや山間，ビルの谷間など電波の届かない場所では受信が途絶える。

メモリ容量の問題については，本稿では差分ファイルの編集指令に従って旧バイナリを直接編集して新バイナリを生成する，インプレース差分更新を bspatch に導入する。

不安定な電源の問題については，更新処理のコンテキストを定期的に不揮発メモリに保存し，不意の給電停止の後でも更新処理を正しく再開できるようにすることで対応する。

不安定な受信環境の問題については本稿では触れず，今後の研究報告に譲る。この問題はファームウェアの差分更新の問題ではなく，むしろ放送受信時のバッファ管理の問題である。

4.2 インプレース差分更新

bsupdate のオリジナルの実装では，差分ファイルの編集命令に従って古いバイナリを編集した結果，すなわち新しいバイナリをバッファ上に構築する。したがって，古いバイナリ，

Vol.2011-SLDM-149 No.23
Vol.2011-EMB-20 No.23
2011/3/18

新しいバイナリ，差分ファイルの大きさをそれぞれ u，v，w とすれば，$u+v+w+O(1)$ ものメモリ量が必要である。

古いバイナリを直接編集して新しいバイナリを生成する更新方式をインプレース更新と呼び，これまでにいくつかのインプレース型差分更新アルゴリズムが提案されている[9)–11)]。本稿では，文献[9)] で提案された Conflicting Read Write Interval グラフを用いた手法を簡略化して応用し，`bsupdate` におけるインプレース更新を実現する。インプレース型 `bsupdate` における所要メモリ量は $\max(u,v)+w+O(1)$ となる。

`bsupdate` においてインプレース更新を実現する際に問題となるのは，後の編集指令で読み込まれる予定の古いバイナリファイル上のメモリ領域を，前の編集指令の Copy & Modify 処理，または New 処理が先に更新する,「書込み干渉（write confliction）」の問題が生じることである。そこで編集指令の実行順序を変更し書込み干渉の問題を解決する。

書込み干渉を生じない編集指令の実行順序を決定するにあたって書込み干渉グラフ G を生成する。書込み干渉グラフの節点は，編集指令の Copy & Modify 処理および New 処理と一対一対応する。節点 u に相当する処理で書き込むメモリ領域の全部または一部を，節点 $v(\neq u)$ に相当する処理で読み込むときに限り，u から v へ向かう枝を設ける。紙面の都合証明は割愛するが，G が循環グラフとなることはない。G の節点をトポロジカルソートした逆順に編集指令の処理を施すと書込み干渉は生じない。但し，古いバイナリの読出し領域と新しいバイナリにおける書込み領域に重なりがある Copy & Modify 処理は注意が必要である。読出し領域が書込み領域より前にある場合は，読出し領域の末尾要素から先頭要素に向かう順番で読出し領域の Copy & Modify を行う「逆方向複写」をしなければならない。逆に，読出し領域が書込み領域より後にある場合は，読出し領域の先頭要素から末尾要素に向かう順番で読出し領域の Copy & Modify を行う「順方向複写」をしなければならない。

図 2 の例を考える。旧バイナリ中の C_i は i 番目の編集指令の Copy & Modify 処理で読み出されるメモリ領域，S_i は同じく Skip 処理で読み飛ばされるメモリ領域である。新バイナリ中の C_i は i 番目の編集指令の Copy & Modify 処理で書き込まれるメモリ領域，N_i は同じく New 処理で書き込まれるメモリ領域である。両バイナリを比較することで図右の書込み干渉グラフが生成される。グラフの節点の右下の数字はこのグラフをトポロジカルソートした一結果を逆順にしたときの番号である。この番号の順番で，すなわち C_1，C_3，C_2，N_1，C_4，N_3，C_5，N_5，C_6 の順番で編集指令の処理を行うと書込み干渉の問題が生じない。但し，Copy & Modify 処理について，C_5，C_6 は順方向複写を，C_2，C_3，C_4 は逆方向複写を行わなければならず，C_1 は順方向複写でも逆方向複写でもどちらでも構わない。

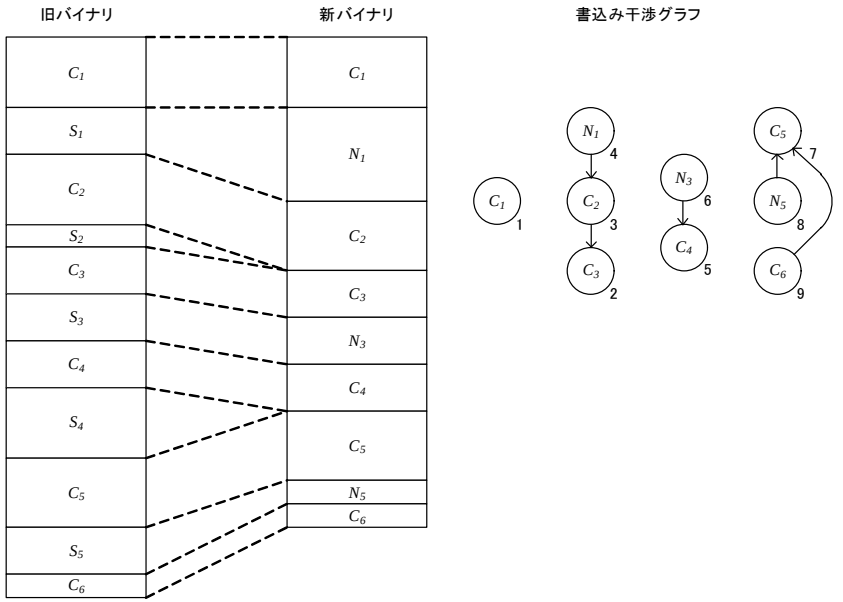

図 2 インプレース更新

4.3 不測の給電停止に耐えるフラッシュメモリ更新手順

一般的に車載機器への給電はエンジン停止によって途絶える。ファームウェア更新中に給電が止まった場合であってもファームウェアを最後まで正しく更新するために，ファームウェア更新に関するコンテキストを定期的に不揮発メモリに保存する。ファームウェア更新中に給電が止まりその後再開されたときには，最後に保存したコンテキストに基づいて更新処理を再開する。

ここで今日，ファームウェアは NOR 型フラッシュメモリに保存されていることを考慮しなければならない。NOR 型フラッシュメモリに書込みを行う場合，ブロック単位でデータを消去した後に書込みを行わなければならない。そこで，Copy & Modify 処理，または New 処理（以下「編集処理」と呼ぶ）をブロック単位に実施する。あるひとつの編集処理の書込み対象となるメモリ領域が複数のブロックにまたがる場合，すべてのブロックを処理し終える前に給電が停止したときに，その編集処理を正しく再開できるように編集処理のコンテキストも保存する必要がある。

Vol.2011-SLDM-149 No.23
Vol.2011-EMB-20 No.23
2011/3/18

更新処理（ならびに更新処理を構成する編集処理）のコンテキストの内容は以下の通りである。但し，∗印は Copy & Modify 処理の場合のみ有効な要素である。

- 現在実行している編集処理の番号
- 編集処理のコンテキスト
  - 編集処理のフェーズ
  - 書込み領域の先頭番地
  - 読出し領域の先頭番地 *
  - 複写方向 *
  - 書込みバイト数
  - 書込み完了バイト数

更新処理のコンテキストはフラッシュメモリ，またはバッテリバックアップされたメモリなど不揮発メモリ上に保存する。

更新処理は次の手順で実行する。

(1) 現在実行している編集処理の番号を最初の編集処理の番号に設定，編集処理の実行フェーズを「ブロック待避」として，コンテキストを不揮発メモリに保存する。
(2) 編集処理がこれから更新しようとするフラッシュメモリのブロック τ を特定し，τ の内容を不揮発メモリ上の待避用ブロックにコピーする。
(3) 編集処理の実行フェーズを「ブロック構築」として，コンテキストを不揮発メモリに保存する。
(4) τ を消去する。
(5) 編集処理の対象とならないメモリ領域の内容を 2 で待避した待避用ブロックから回復する。
(6) 編集処理の実行，すなわち τ に対する書込みを実施する。編集処理が τ に属する番地を読み出さなければならない場合，τ はすでに消去されているため，代わりに 2 で待避した待避用ブロックから読み出す。
(7) 次の編集処理の準備をする。
  (a) 編集処理を最後まで実行した場合（＝未処理のブロックがない場合）は，現在実行している編集処理の番号を次の編集処理の番号に設定，編集処理の実行フェーズを「ブロック待避」としてコンテキストを不揮発メモリに保存する。
  (b) そうでない場合は，書込み完了バイト数を更新，編集処理の実行フェーズを「ブロック待避」としてコンテキストを不揮発メモリに保存する。
(8) 2 に戻る。

中断された更新処理の再開は，不揮発メモリに保存されたコンテキストに基づいて行う。コンテキスト中の編集処理の実行フェーズが「ブロック待避」ならば上記の 2 から,「ブロック構築」ならば 4 から更新処理を再開する。

4.4 シャドウ RAM 構成の場合の更新処理の簡単化

フラッシュメモリ上のバイナリを直接実行するのではなく，システム起動時にファームウェアを RAM に転送してから実行するようにしている，いわゆるシャドウ RAM 構成のシステムでは，上に述べた差分更新処理を大幅に簡略化できる。

放送を通じて得られた差分ファイルはフラッシュメモリに格納するのみとし，フラッシュメモリに既納の古いバイナリは一切変更しない。システム起動時にはフラッシュメモリからバイナリを転送する際は，バイナリをそのまま転送するのではなく，フラッシュメモリ上のバイナリを差分ファイルの編集指令に基づいて編集し，編集結果，すなわち新しいバイナリを RAM 上に展開する。

5. ま と め

本稿では，放送による車載機器向けソフトウェア差分更新方式について述べた。提案方式において，放送局はバイナリファイル差分抽出ツールである `bsdiff` を用いて新旧ファームウェアの差分を求めて放送し，車載機器は同差分を受信して自身の持つ古いファームウェアを新しいものに書き換える。このように，新しいファームウェア全体ではなく新旧ファームウェアの差分を放送することで，放送の帯域使用量と車載機器における受信バッファ用の要求メモリ量を削減した。車載機器側でのファームウェアの更新は，`bsdiff` の生成した差分を適用する `bsupdate` を改造したものを用いて行った。新ファームウェアをバッファに構築するのではなく，旧ファームウェアを直接編集して新ファームウェアを構築するインプレース更新により更新時のメモリ消費量を抑えた。また，ファームウェア更新中に定期的に更新処理のコンテキストを保存することにより，エンジン停止等によりファームウェア更新中に給電が止まっても，給電再開後にファームウェア更新を正しく再開できるようにするフラッシュメモリ更新手順を提案した。さらに，シャドウ RAM 構成のシステムにおいては，フラッシュメモリの内容を RAM に転送する際に差分を適用することが可能であり，インプレース更新を行う必要はなく，更新処理を大幅に簡単化できることを指摘した。

本稿で提案したフラッシュメモリ更新手順は同一のブロックを複数回消去したり，同一番地のデータを複数回書き込む可能性がある。NOR 型フラッシュメモリは書込みに時間を要

Vol.2011-SLDM-149 No.23
Vol.2011-EMB-20 No.23
2011/3/18

するうえに，原理上書込み可能回数に制限があるため，ブロック消去や二度書きは可能な限り削減することが望ましい。今後の重要な課題としてフラッシュメモリ更新手順の効率化が挙げられる。

また，本稿では放送される差分ファイルの受信については検討していなかったが，不安定な受信環境下での差分ファイル全体を確実に受信したことを検証する手段，ならびに受信時のバッファ管理についても今後検討を加えたい。

謝辞

本研究は科研費（課題番号 21700035）の助成を受けたものである。

参 考 文 献

1) Manfred Broy, “Challenges in Automotive Software Engineering,” *Proc. the 28th Int. Conf. on Software Engineering (ICSE2006)*, pp.33–40, May 2006.
2) Michael Hicks, “Dynamic Software Updating,” *ACM Transactions on Programming Languages and System*, Vol.27, No.6, pp.1049–1096, Nov. 2005.
3) 清原 良三, 「組込みソフトウェア向けバイナリー差分抽出方式」, 電子情報通信学会論文誌, VOL.J90-D, No.6, pp.1375–1382, 2007 年 6 月.
4) World Wide Web Consortium, `http://www.w3.org/TR/NOTE-gdiff-19970901`
5) 清原 良三, 「携帯電話のソフトウェアアドインに関する研究」, 大阪大学大学院情報科学研究科博士論文, 2008.
6) David Salomon, *Data Compression Complete Reference 4th Ed.*, Springer-Verlag, pp. 850–940, 2007.
7) Dimitre Trendafilov, “zdelta: An Efficient Delta Compression Tool,” *Technical Report TR-CIS-2002-02*, Department of Computer and Information Science, Polytechnic University, June 2002.
8) Colin Percival, “Naïve Differences of Executable Code,” `http://www.daemonology.net/bsdiff`, 2003.
9) R. C. Burms and D. D. E. Long, “In-Place Reconstruction of Delta Compressed File,” *ACM Symp. on Principles of Distributed Computing*, pp.267–275, Jul. 1998.
10) Dana Shapira and James A. Storer, “In Place Differential File Compression,” *The Computer Journal Advance Access published*, Vol.48, pp.667–691, Aug. 2005.
11) Yun Chan Cho and Jae Wook Jeon, “In-Place Reconstructible Delta Compression Using Alleviated Greedy Matching Algorithm,” *Proc. Int. Conf. on Industrial Informatics (INDIN2008)*, pp.1596–1601, Jul. 2008.